O'REILLY®

Включая Xcode 5

# iOS

# ПРИЕМЫ ПРОГРАММИРОВАНИЯ

ПИТЕР®

Вандад Нахавандипур

Vandad Nahavandipoor

# iOS 7 Programming Cookbook

Beijing • Cambridge • Farnham • Köln • Sebastopol • Tokyo O'REILLY®

Вандад Нахавандипур

# iOS
# ПРИЕМЫ
# ПРОГРАММИРОВАНИЯ

ПИТЕР®

Москва · Санкт-Петербург · Нижний Новгород · Воронеж
Ростов-на-Дону · Екатеринбург · Самара · Новосибирск
Киев · Харьков · Минск
2014

*В. Нахавандипур*

# iOS. Приемы программирования

*Серия «Бестселлеры O'Reilly»*

Перевел с английского *О. Сивченко*


| | |
|---|---|
| Заведующий редакцией | *Д. Виницкий* |
| Ведущий редактор | *Н. Гринчик* |
| Научный редактор | *О. Сивченко* |
| Литературный редактор | *Н. Рощина* |
| Художник | *Л. Адуевская* |
| Корректор | *Е. Павлович* |
| Верстка | *А. Барцевич* |

ББК 32.973.2-018.1
УДК 004.43

**Нахавандипур В.**

Н34 iOS. Приемы программирования. — СПб.: Питер, 2014. — 832 с.: ил. — (Серия «Бестселлеры O'Reilly»).

ISBN 978-5-496-01016-0



Книга, которую вы держите в руках, представляет собой новый, полностью переписанный сборник приемов программирования по работе с iOS. Он поможет вам справиться с наболевшими проблемами, с которыми приходится сталкиваться при разработке приложений для iPhone, iPad и iPod Touch. Вы быстро освоите всю информацию, необходимую для начала работы с iOS 7 SDK, в частности познакомитесь с решениями для добавления в ваши приложения реалистичной физики или движений — в этом вам помогут API UIKit Dynamics.

Вы изучите новые многочисленные способы хранения и защиты данных, отправки и получения уведомлений, улучшения и анимации графики, управления файлами и каталогами, а также рассмотрите многие другие темы. При описании каждого приема программирования приводятся образцы кода, которые вы можете смело использовать.


**12+** (В соответствии с Федеральным законом от 29 декабря 2010 г. № 436-ФЗ.)


ISBN 978-1449372422 англ.





ISBN 978-5-496-01016-0





ООО «Питер Пресс», 192102, Санкт-Петербург, ул. Андреевская (д. Волкова), 3, литер А, пом. 7Н.

Налоговая льгота — общероссийский классификатор продукции ОК 005-93, том 2; 95 3005 — литература учебная.

Подписано в печать 05.05.14. Формат 70×100/16. Усл. п. л. 67,080. Тираж 500. Заказ 0000.

Отпечатано в полном соответствии с качеством предоставленных издательством материалов в Чеховский Печатный Двор. 142300, Чехов, Московская область, г. Чехов, ул. Полиграфистов, 1.

# Краткое содержание

# Оглавление

# Предисловие

Это издание книги является не просто дополненной, а полностью переработанной версией предыдущего. В iOS 7 изменилось все: внешний вид и функциональная сторона операционной системы, способы использования наших устройств с iOS и, самое главное, принципы программирования для таких устройств. Действительно, без серьезной переработки всей книги было не обойтись. Я добавил в нее примерно 50 новых разделов-рецептов, затронув в них такие вопросы, как динамика UIKit, работа с видами-коллекциями, связкой ключей, удаленными уведомлениями и пр. Кроме того, я проработал все примеры кода и иллюстрации и обновил их с учетом iOS 7.

iOS 7 — огромный шаг вперед в развитии той операционной системы, которую любим все мы, пользователи и разработчики. Нам нравится как работать с нею, так и программировать для нее. Возможно, вы заметили, какое внимание в iOS 7 уделяется *динамичности* системы: ваш пользовательский интерфейс должен реагировать на различные движения и перемещения, которые могут происходить с устройством. Я имею в виду следующее: Apple стремится, чтобы разработчики по-настоящему внимательно относились к деталям создаваемых приложений, обогащали их реалистичной физикой и динамикой. Именно поэтому Apple дополнила iOS SDK новым элементом — UIKit Dynamics, а в этой книге данной концепции посвящена целая глава. iPhone становится все более высокотехнологичным устройством, оставаясь при этом довольно дорогим. Соответственно, запросы его пользователей также растут. И это понятно. Пользователь только что приобрел совершенно фантастический новейший iPhone или iPad и хочет найти на нем замечательные приложения, максимально полно и эффективно задействующие все возможности этих устройств.

Именно поэтому сейчас разработчик как никогда нуждается в глубоких знаниях SDK, чтобы понимать, что этот SDK может предложить программисту для создания более классных и быстрых приложений. Apple реализовала в SDK для iOS 7 множество классных новых API, и в этой книге мы подробно с ними познакомимся.

Главной особенностью iOS 7 является динамика!

Прежде чем вы приступите к изучению этой книги, я хотел бы немного рассказать о своем профессиональном опыте и о том, чем смогу помочь вам в путешествии по ее страницам. Поведаю, что я за человек и как началась моя большая любовь к iOS. Еще ребенком я впервые попробовал писать код на Basic, тогда у меня был компьютер Commodore 64. Потом я купил себе ПК и принялся экспериментировать с кодом на ассемблере. Сначала это был 8-битный ассемблер для DOS. Затем я попробовал в домашних условиях написать собственную операционную систему,

которая так и не была выпущена в качестве коммерческой программы. Она предназначалась для работы на 32-битной архитектуре на процессоре Intel x86.

Среди языков программирования, в которых я пробовал свои силы, особое место занимают ассемблер и Objective-C. Они мне по-настоящему нравились и очень отличались от всех остальных. В ассемблере меня привлекала его чистота: каждая команда делает всего одну вещь, и делает ее хорошо. Думаю, Objective-C приглянулся мне по схожей причине. На самом деле этот признак ассемблера и Objective-C прослеживается во всей операционной системе iOS. Конечно, iOS — это именно операционная система, а ни в коем случае не язык программирования, но во всем, что она делает, iOS обходит своих конкурентов. Это касается как ее простоты, так и того чистого потенциала, которым она наделяет вас, комбинируя аппаратные и программные возможности. В iOS используются замечательные собственные технологии, в частности GCD. Эта операционная система задает такую высокую планку в области удобства и простоты использования, которая до сих пор остается беспрецедентной.

Все разделы всех глав этой книги полностью обновлены с учетом iOS 7. Кроме того, обновлены все скриншоты, а также появилось много новых разделов. В частности, вы найдете их в темах, посвященных связке ключей, динамике пользовательского интерфейса, представлениям для работы с коллекциями, удаленным и локальным уведомлениям, и во многих других. Они были написаны специально для этого издания книги. Я писал новое издание с огромным интересом, и, надеюсь, с не меньшим интересом вы будете знакомиться с новыми возможностями, которые в ней рассмотрены. Вероятно, это будет очень ценное дополнение к вашей технической библиотеке.

## Для кого предназначена книга

Предполагается, что читатель хорошо знаком со средой для разработки в iOS и знает, как создать приложение для iPhone или iPad. Эта книга не подойдет программисту-новичку в качестве вводного пособия, но в ней описаны удобные способы решения конкретных задач, с которыми могут столкнуться программисты разного уровня — и новички, и эксперты.

## Как построено издание

В этой книге мы рассмотрим фреймворки и классы, доступные в SDK для iOS 7. Я приложил все возможные усилия, чтобы на ее страницах научить вас работе с новейшими и самыми классными API iOS. Разумеется, некоторые пользователи ваших приложений могут работать и с более старыми версиями iOS. Пожалуйста, не забывайте об этих пользователях и выбирайте API рационально, с учетом минимальной версии iOS, на работу с которой рассчитано ваше приложение.

Apple рекомендует писать приложения таким образом, чтобы они поддерживались и работали в версиях iOS 6 и 7. Таким образом, в качестве базового SDK вы должны применять последнюю версию этого инструментария (имеется в виду SDK,

на базе которого будет компилироваться ваше приложение) и выбирать в качестве целевой платформы iOS 6 (если это не противоречит поставленным перед вами бизнес-требованиям). Если от вас требуется написать такое приложение, которое должно поддерживаться только в iOS 7, вас ждет масса интересного. Ведь вы будете работать с превосходными API, появившимися только в iOS 7 и рассмотренными в этой книге.

Рассмотрим краткое содержание материала.

- **Глава 1. Реализация контроллеров и видов.** В этой главе объясняется структура классов в Objective-C и рассматриваются способы реализации объектов. Здесь мы поговорим о свойствах и делегатах, а также о подписке по ключам и индексам. Даже если вы хорошо разбираетесь в Objective-C, настоятельно рекомендую вам изучить данную главу, пусть даже бегло. Это нужно, чтобы понять базовый материал, на котором построены остальные главы. В этой главе мы также рассмотрим типичные способы обращения с различными элементами пользовательского интерфейса — видами-предупреждениями, сегментированными элементами управления, переключателями и надписями. Также поговорим о настройке этих компонентов с применением новейших API, имеющихся в SDK.
- **Глава 2. Создание динамических и интерактивных пользовательских интерфейсов.** Глава рассказывает о UIKit Dynamics — новейшем дополнении, появившемся во фреймворке UIKit. Эти динамические компоненты позволяют обогащать компоненты вашего пользовательского интерфейса реалистичной физикой и динамикой. Интерфейсы получатся еще более живыми, причем с меньшими усилиями с вашей стороны.
- **Глава 3. Автоматическая компоновка и язык визуального форматирования.** В этой главе вы узнаете, как пользоваться возможностью автоматической компоновки в iOS SDK и создавать пользовательские интерфейсы таким образом, чтобы их можно было гибко масштабировать (сжимать и растягивать) на экране практически в любом направлении.
- **Глава 4. Создание и использование табличных видов.** В этой главе рассказано, как работать с табличными видами, чтобы создавать приложения iOS, производящие впечатление профессионально выполненной работы. По природе табличные виды очень динамичны, поэтому программисту иногда бывает сложно понять, как с ними работать. Прочитав эту главу, изучив и опробовав на практике код из приведенных примеров, вы научитесь удобным приемам работы с табличными видами.
- **Глава 5. Выстраивание сложных макетов с помощью сборных видов.** Виды-коллекции уже довольно давно вошли в арсенал программистов, работающих с OS X. Теперь Apple решила включить те же самые API в iOS SDK, предоставив их, таким образом, iOS-программистам. Виды-коллекции во многом напоминают табличные виды, но отличаются значительно более широкими возможностями конфигурирования и динамичностью. Если в табличных видах мы имеем дело с концепцией разделов, каждый из которых делится на строки, в видах-коллекциях появляются также столбцы. Поэтому в виде-коллекции вы при желании можете отображать много элементов в одной строке. В этой главе мы

рассмотрим все великолепные пользовательские интерфейсы, которые можно создавать с применением видов-коллекций.

- **Глава 6. Раскадровки.** Здесь мы поговорим о процессе раскадровки. Это новый способ определения связей между различными экранами (видами), задействованными в приложении. Самая приятная особенность раскадровки заключается в том, что вам совсем не обязательно вообще что-то знать о программировании для iOS, чтобы написать и запустить простое приложение. Это свойство очень помогает специалистам, работающим вне команды, — аналитикам, владельцам продукта или дизайнерам, — а также команде разработчиков познакомиться со свойствами, которыми обладают компоненты пользовательского интерфейса в iOS, и, имея такие знания, создавать более надежные продукты. Кроме того, преимущества, которые дает раскадровка, облегчают работу программиста на этапе прототипирования. Раскадровка — это просто интересное дело, независимо от того, занимаетесь ли вы ею на бумаге или с помощью Xcode.
- **Глава 7. Параллелизм.** Человек умеет делать одновременно несколько вещей, причем особенно не задумываясь о том, как это происходит. С развитием информационных технологий мобильные устройства также становятся многозадачными. Разработчику, пишущему программы для таких устройств, предоставляются инструменты и механизмы, которые позволяют выполнять несколько задач в определенный момент времени. Этот феномен называется параллелизмом или конкурентным программированием. В главе 5 вы узнаете о технологии Grand Central Dispatch, с помощью которой Apple в основном обеспечивает параллелизм в iOS. В этой главе мы поговорим также о таймерах, потоках и операциях.
- **Глава 8. Безопасность.** iOS 7 — весьма безопасная операционная система. Приложения, которые мы для нее пишем, также должны соответствовать определенным стандартам и практикам обеспечения безопасности. В этой главе будет рассмотрено, как пользоваться преимуществами различных API связки ключей и повысить безопасность ваших приложений. Мы обсудим также различные меры, помогающие повысить безопасность вашего пользовательского интерфейса.
- **Глава 9. Core Location и карты.** В этой главе обсуждается работа с комплектом для программирования карт (Map Kit) и основных геолокационных API — то есть речь пойдет о написании приложений для iOS, располагающих информацией о местоположении устройства. Сначала поговорим о картах, потом обсудим, как определяется местоположение устройства, и снабдим ваши карты пользовательскими аннотациями. Потом изучим геокодирование и обратное геокодирование, а также методы, входящие в состав фреймворка Core Location, доступные лишь в версии iOS 7 SDK.
- **Глава 10. Реализация распознавания жестов.** Здесь демонстрируется использование механизмов, распознающих жесты пользователя на сенсорном экране. Они позволяют пользователю легко обращаться с графическим интерфейсом ваших приложений для iOS. В этой главе вы научитесь применять соответствующие механизмы, доступные в SDK для iOS, а также изучите рабочие примеры, которые протестированы в операционной системе iOS 7.

- **Глава 11. Сетевые функции, JSON, XML и Twitter.** Глава рассказывает о встроенных синтаксических анализаторах для JSON и XML. На базе этой информации в главе рассматриваются различные API для сетевых взаимодействий, а также обсуждается, как вы можете реализовать в ваших приложениях функции для общения в социальных сетях. С помощью таких возможностей пользователи смогут обмениваться своими данными и творчеством на подобных ресурсах, например в Facebook.
- **Глава 12. Управление файлами и каталогами.** Одна из самых важных задач, которые нам как разработчикам приходится решать при программировании для iOS, — это управление файлами и каталогами. В этой главе речь пойдет о создании, считывании, записи и удалении файлов. Здесь вы найдете информацию, достаточно подробную для того, чтобы наладить управление файлами и каталогами с помощью средств iOS SDK.
- **Глава 13. Камера и библиотека фотографий.** В данной главе показано, как обнаружить доступность камеры на передней и задней поверхностях устройства с системой iOS. Кроме того, вы научитесь обращаться к библиотеке фотографий посредством фреймворка ресурсов (Assets Framework). В конце главы рассказано о том, как редактировать видео прямо на устройстве с iOS с помощью встроенного контроллера видов.
- **Глава 14. Многозадачность.** В этой главе показано, как создавать приложения, ориентированные на многозадачность и хорошо функционирующие на устройствах с iOS. Вы узнаете об организации фоновых процессов, о воспроизведении аудио и определении местонахождения пользователя в фоновом режиме. Кроме того, мы поговорим о загрузке содержимого по URL, пока ваше приложение находится в фоновом режиме. Опираясь на эту информацию, мы также исследуем некоторые новые API для многозадачности, появившиеся в iOS 7. Они позволяют нам скачивать контент периодически, пока приложение работает в фоновом режиме и даже когда оно не запущено.
- **Глава 15. Уведомления.** Уведомления — это объекты, несущие определенную информацию, которая может передаваться множеству получателей методом широковещания. В этой главе мы обсудим уведомления, в том числе локальные и пуш-уведомления. Вы узнаете, как использовать новейшие возможности, встроенные в Xcode, и легко включать эти функции в свои приложения.
- **Глава 16. Фреймворк Core Data.** Глава описывает детали стеков Core Data и демонстрирует, из чего состоят эти стеки. Изучив ее, вы сможете проектировать собственные объектно-ориентированные модели данных прямо в Xcode с помощью редактора моделей Core Data, а также создавать и получать объекты в Core Data. Опираясь на эту информацию, вы научитесь добавлять в Core Data собственные данные и искать данные с помощью фонового потока. В таком случае ваш поток пользовательского интерфейса останется свободен и сможет своевременно обрабатывать пользовательские события.
- **Глава 17. Графика и анимация.** В этой главе дается введение во фреймворк Core Graphics. Вы узнаете, как отрисовывать изображения и текст в графическом контексте, рисовать линии, прямоугольники и пути, а также многое другое.

Также вы познакомитесь с новыми API из iOS SDK, позволяющими фиксировать содержимое ваших видов в форме скриншотов.

- **Глава 18. Фреймворк Core Motion.** Как следует из названия, глава посвящена рассмотрению фреймворка Core Motion. С помощью Core Motion вы получаете доступ к акселерометру и гироскопу, установленным на устройстве с iOS. Кроме того, вы сможете регистрировать встряхивание устройства. Разумеется, не на всех устройствах с iOS имеются акселерометр и гироскоп, поэтому вы также узнаете, как определять доступность этого оборудования.
- **Глава 19. Фреймворк Pass Kit.** В этой главе описан Passbook — виртуальный кошелек, который позволяет управлять вашими купонами, посадочными талонами, железнодорожными и автобусными билетами, а также другими подобными документами. Здесь вы узнаете все необходимое для создания ваших собственных путевых документов с цифровой подписью и научитесь с легкостью раздавать их пользователям.

# Дополнительные ресурсы

Время от времени я обращаюсь к официальной документации Apple. Некоторые описания с сайта Apple точны, и пытаться изложить их своими словами — все равно что изобретать велосипед. Здесь я перечислил наиболее важные официальные документы и руководства, предлагаемые Apple, которые должен прочитать каждый разработчик, профессионально пишущий программы для iOS.

Для начала предлагаю ознакомиться с iOS Human Interface Guidelines (Руководством по созданию пользовательских интерфейсов для всех устройств iOS) (https://developer.apple.com/library/ios/documentation/UserExperience/Conceptual/MobileHIG/Introduction/Introduction.html). Этот документ необходимо прочитать любому программисту, работающему с iOS. На самом деле я считаю, что эти документы обязательно должны прочитать также сотрудники отделов разработки и дизайна продукции в любой компании, занимающейся iOS.

Рекомендую также просмотреть документ iOS Application Programming Guide (Руководство по программированию приложений для iOS), имеющийся в iOS Reference Library (Справочной библиотеке iOS), где даются полезные советы по созданию отличных приложений для iOS: https://developer.apple.com/library/ios/#documentation/iPhone/Conceptual/iPhoneOSProgrammingGuide/Introduction/Introduction.html.

В iOS 7 значительно изменились принципы представления компонентов пользовательского интерфейса на экране. Мы подробно обсудим все эти изменения и поговорим о том, как программист может применять эти новейшие API для создания замечательных приложений для iOS 7. Кроме того, я рекомендую вам ознакомиться с документом iOS 7 UI Transition Guide (Руководство по переходу на пользовательский интерфейс iOS 7) от Apple (https://developer.apple.com/library/ios/documentation/UserExperience/Conceptual/TransitionGuide/index.html#//apple_ref/doc/uid/TP40013174). В нем описаны все изменения, связанные с пользовательским интерфейсом, появившиеся в последней версии SDK.

Читая главу 12, вы обратите внимание на то, что в ней одной из наиболее важных тем являются блоковые объекты. Блоковые объекты рассматриваются в книге вкратце, но чтобы подробнее разобраться с этой темой, рекомендую познакомиться с руководством A Short Practical Guide to Blocks (Краткое практическое руководство по блоковым объектам), доступным по следующей ссылке: https://developer.apple.com/library/ios/#featuredarticles/Short_Practical_Guide_Blocks/index.html%23/apple_ref/doc/uid/TP400 09758.

В книге я буду часто упоминать пакеты (bundles) и говорить о том, как загружать из пакетов изображения и данные. В издании будет кратко рассказано о пакетах, но если хотите разобраться с ними подробнее, прочтите Bundle Programming Guide (Руководство по программированию пакетов) по адресу: https://developer.apple.com/library/ios/#documentation/CoreFoundation/Conceptual/CFBundles/Introduction/Introduction.html.

# Условные сокращения, используемые в данной книге

В данной книге применяются следующие условные обозначения.

Шрифт для названий

Используется для обозначения URL, адресов электронной почты, а также сочетаний клавиш и названий элементов интерфейса.

`Шрифт для команд`

Применяется для обозначения программных элементов — переменных и названий функций, типов данных, переменных окружения, операторов, ключевых слов и т. д.

```
Шрифт для листингов
```

Используется в листингах программного кода.

Данный символ означает совет, замечание практического характера или общее замечание.

Данный символ означает предостережение.

# Работа с примерами кода

Эта книга написана для того, чтобы помочь вам в работе. В принципе, вы можете использовать код, содержащийся в этой книге, в ваших программах и документации. Можете не связываться с нами и не спрашивать разрешения, если собираетесь воспользоваться небольшим фрагментом кода. Например, если вы пишете программу и кое-где вставляете в нее код из этой книги, разрешения не требуется. Однако если вы запишете на диск примеры из книг издательства O'Reilly и начнете раздавать

или продавать такие диски, на это необходимо получить разрешение. Если вы цитируете эту книгу, отвечая на вопрос, или воспроизводите код из нее в качестве примера, на это не требуется разрешения. Если вы включаете значительный фрагмент кода из этой книги в документацию по вашему продукту, на это требуется разрешение.

## Нам интересны ваши отзывы

Все примеры кода из этой книги были протестированы на iPhone 4, iPhone 3GS и эмуляторе iPhone/iPad, но не исключено, что у вас все же возникнут какие-то сложности. Например, у вас будет иная версия SDK, нежели та, в которой компилировался и тестировался код из примера. Информация, изложенная в этой книге, проверялась на каждом этапе подготовки издания. Тем не менее мы могли допустить какие-то ошибки или чего-то недосмотреть, поэтому с благодарностью примем от вас информацию о любых подобных недочетах, которые могут вам встретиться, а также все ваши предложения о том, как можно было бы улучшить будущие издания книги. С автором и редакторами можно связаться по следующему адресу:

O'Reilly Media, Inc.
1005 Gravenstein Highway North
Sebastopol, CA 95472
(800) 998-9938 (в США или Канаде)
(707) 829-0515 (международный или местный телефон)
(707) 829-0104 (факс)

## Благодарности

Энди Орам, мой любезный редактор, вновь потрудился на славу и внимательно проработал все изменения, появившиеся в новом издании книги. Фактически эта книга переработана полностью, это касается и содержащихся в ней скриншотов и примеров кода. Я хотел бы поблагодарить также Кшиштофа Гробельного и Кшиштофа Гутовского — моих хороших друзей и коллег, выполнивших техническое рецензирование книги. Без их участия она ни за что не оказалась бы в ваших руках.

Особой благодарности заслуживает Рэйчел Румелиотис, поддерживавшая меня и Энди. В первую очередь спасибо ей за ту административную работу, которая на первый взгляд как будто не видна. Кроме того, с наилучшей стороны себя показала Меган Конноли из издательства O'Reilly. Она терпеливо сносила мои причитания о бумажной работе, сотрудничество с ней доставило одно удовольствие. Благодарю Джессику Хозман за то, что помогла нам справиться с проблемами, которые возникали с Git. Я и поверить не мог, что те простые решения, которые она мне подсказывала, действительно сработают. Но они работали, а я порой чувствовал себя идиотом.

Последние, но немаловажные благодарности хочется высказать Алине Риззони, Бруно Пэкхему и Томасу Пэкхему за их преданную дружбу. Я счастлив, что знаю их, и высоко ценю их помощь и поддержку.

# От издательства

Ваши замечания, предложения и вопросы отправляйте по адресу электронной почты vinitski@minsk.piter.com (издательство «Питер», компьютерная редакция).

Мы будем рады узнать ваше мнение!

На сайте издательства http://www.piter.com вы найдете подробную информацию о наших книгах.

# 1 Реализация контроллеров и видов

## 1.0. Введение

В iOS 7 появилось множество новых пользовательских возможностей, а также масса новых API, с которыми мы, программисты, можем вволю экспериментировать. Вероятно, вы уже знаете, что в iOS 7 разительно изменился пользовательский интерфейс. Во всех предыдущих версиях он оставался практически неизменным по сравнению с первой версией iOS, и поэтому многие приложения разрабатывались так, как будто пользовательский интерфейс никогда не изменится. В настоящее время графические дизайнеры столкнулись с целым букетом проблем, так как теперь требуется создавать интерфейсы и продумывать пользовательские взаимодействия с программой так, чтобы программа хорошо смотрелась и в iOS 7, и в более ранних версиях.

Чтобы программировать приложения для iOS 7, вы должны знать основы языка Objective-C, с которым мы будем работать на протяжении всей этой книги. Как понятно из названия, язык Objective-C основан на C, но имеет определенные расширения, которые облегчают оперирование объектами. Объекты и классы имеют фундаментальное значение в объектно-ориентированном программировании (ООП). К числу объектно-ориентированных языков относятся Objective-C, Java, C++ и многие другие. В Objective-C, как и в любом объектно-ориентированном языке, вы имеете доступ не только к объектам, но и к примитивам. Например, число –20 (минус двадцать) можно выразить в виде примитива следующим образом:

```
NSInteger myNumber = -20;
```

В этой простой строке кода определяется переменная `myNumber`, относящаяся к типу данных `NSInteger`. Ее значение устанавливается в 20. Так определяются переменные в языке Objective-C. Переменная — это простое присваивание имени местоположению в памяти. В таком случае если мы задаем 20 в качестве значения переменной `myNumber`, то сообщаем машине, что собираемся выполнить фрагмент кода, который поместит указанное значение в область памяти, соответствующую переменной `myNumber`.

В сущности, все приложения iOS используют архитектуру «модель — вид — контроллер» (MVC). С архитектурной точки зрения модель, вид и контроллер — это три основные составляющие приложения iOS.

*Модель* — это мозг приложения. Она выполняет все вычисления и создает для себя виртуальный мир, в котором может существовать сама, без видов и контроллеров. Иными словами, вы можете считать модель виртуальной копией вашего приложения, без интерфейса.

*Вид* — это окно, через которое пользователь взаимодействует с вашим приложением. В большинстве случаев вид отображает содержимое модели, но, кроме того, он же воспринимает и действия пользователя. Любые контакты между пользователем и вашим приложением отправляются в вид. После этого они могут быть перехвачены контроллером вида и переданы в модель.

*Контроллеры* в программах iOS — это, как правило, контроллеры видов, которые я только что упомянул. Контроллер вида является, в сущности, переходным звеном между моделью и видом. Он интерпретирует события, происходящие с одной стороны, и по мере необходимости использует эту информацию для внесения изменений на другой стороне. Например, если пользователь изменяет какое-либо поле в виде, то контроллер гарантирует, что и модель изменится соответствующим образом. А если модель получит новые данные, то контроллер прикажет виду отобразить их.

В этой главе вы узнаете, как выстраивать структуру приложения iOS и использовать виды и контроллеры видов для создания интуитивно понятных приложений.

В этой главе мы будем создавать большинство компонентов пользовательского интерфейса на базе шаблона Single View Application из Xcode. Чтобы воспроизвести приведенные инструкции, следуйте рекомендациям, приведенным в подразделе «Создание и запуск вашего первого приложения для iOS» данного раздела. Убедитесь в том, что ваше приложение является универсальным, а не ориентировано только на iPhone или на iPad. Универсальное приложение может работать как на iPhone, так и на iPad.

## Создание и запуск вашего первого приложения для iOS

Прежде чем подробнее познакомиться с возможностями Objective-C, вкратце рассмотрим, как создать простое приложение для iOS в среде Xcode. Xcode — это интегрированная среда разработки (IDE) для работы с Apple, позволяющая создавать, строить и запускать ваше приложение в эмуляторе iOS и даже на реальных устройствах с iOS. По ходу книги мы подробнее обсудим Xcode и ее возможности, а пока научимся создавать и запускать самое простое приложение. Я полагаю, что вы уже скачали Xcode из Mac App Store и установили ее на своем компьютере. В таком случае выполните следующие шаги.

1. Откройте Xcode, если еще не сделали этого.
2. Выберите в меню пункт File (Файл), далее — New Project (Новый проект).
3. Слева в диалоговом окне создания нового проекта выберите подкатегорию Application (Приложение) в основной категории iOS. Затем справа щелкните на варианте Single View Application (Приложение с единственным видом) и нажмите кнопку Next (Далее).

4. На следующем экране вы увидите поле Product Name (Название продукта). Здесь укажите название, которое будет понятно вам, например `My First iOS App`. В разделе Organization name (Название организации) введите название вашей компании или, если работаете самостоятельно, любое другое осмысленное название. Название организации — довольно важная информация, которую, как правило, придется здесь указывать, но пока она нас не особенно волнует. В поле Company Identifier (Идентификатор компании) запишите `com.mycompany`. Если вы действительно владеете собственной компанией или пишете приложение для фирмы, являющейся вашим работодателем, то замените `mycompany` настоящим названием. Если просто экспериментируете, придумайте какое-нибудь название. В разделе Devices (Устройства) выберите вариант Universal (Универсальное).
5. Как только зададите все эти значения, просто нажмите кнопку Next (Далее).
6. Система предложит сохранить новый проект на диске. Выберите желаемое местоположение проекта и нажмите кнопку Create (Создать).
7. Перед запуском проекта убедитесь, что к компьютеру не подключено ни одного устройства iPhone или iPad/iPod. Это необходимо, поскольку, если к вашему Mac подключено такое устройство, то Xcode попытается запускать приложения именно на устройстве, а не на эмуляторе. В таком случае могут возникнуть некоторые проблемы с профилями инициализации (о них мы поговорим позже). Итак, отключите от компьютера все устройства с системой iOS, а затем нажмите большую кнопку Run (Запуск) в левом верхнем углу Xcode. Если не можете найти кнопку Run, перейдите в меню Product (Продукт) и выберите в меню элемент Run (Запуск).

Ура! Вот и готово простое приложение, работающее в эмуляторе iOS. Может быть, оно и не кажется особенно впечатляющим: в эмуляторе мы видим просто белый экран. Но это лишь первый шаг к освоению огромного iOS SDK. Давайте же отправимся в это непростое путешествие!

## Определение переменных и понятие о них

Во всех современных языках программирования, в том числе в Objective-C, существуют переменные. Переменные — это просто псевдонимы, обозначающие участки (местоположения) в памяти. Каждая переменная может иметь следующие свойства:

- тип данных, представляющий собой либо примитив (например, целое число), либо объект;
- имя;
- значение.

Задавать значение для переменной приходится не всегда, но вы обязаны указывать ее имя и тип. Вот несколько типов данных, которые необходимо знать для написания типичного приложения iOS.

Если тип данных является изменяемым, то вы можете изменить такие данные уже после инициализации. Например, вы можете откорректировать одно из значений в изменяемом массиве, добавлять в него новые значения или удалять их оттуда. Напротив, при работе с неизменяемым типом вы должны предоставлять все значения для него уже на этапе инициализации. Позже нельзя будет пополнить набор этих значений, удалить какие-либо значения или изменить их. Неизменяемые типы полезны в силу своей сравнительно более высокой эффективности. Кроме того, они помогают избежать ошибок, если все значения должны оставаться неизменными на протяжении всего жизненного цикла данных.

- `NSInteger` и `NSUInteger`. Переменные этого типа могут содержать целочисленные значения, например `10`, `20` и т. д. Тип `NSInteger` может содержать как положительные, так и отрицательные значения, но тип `NSUInteger` является беззнаковым, на что указывает буква U в его названии. Не забывайте, что слово «беззнаковый» в терминологии языков программирования означает, что число ни при каких условиях не может быть отрицательным. Отрицательные значения могут содержаться только в числовом типе со знаком.
- `CGFloat`. Содержит числа с плавающей точкой, имеющие десятичные знаки, например `1.31` или `2.40`.
- `NSString`. Позволяет сохранять символьные строки. Такие примеры мы рассмотрим далее.
- `NSNumber`. Позволяет сохранять числа как объекты.
- `id`. Переменные типа `id` могут указывать на объект любого типа. Такие объекты называются *нетипизированными*. Если вы хотите передать объект из одного места в другое, но по какой-то причине не хотите при этом указывать их тип, то вам подойдет именно такой тип данных.
- `NSDictionary` и `NSMutableDictionary`. Это соответственно неизменяемый и изменяемый варианты хеш-таблиц. В хеш-таблице вы можете хранить ключ и ассоциировать этот ключ со значением. Например, ключ `phone_num` может иметь значение `0 5552 4 87700`. Для считывания значений достаточно ссылаться на ассоциированные с ними ключи.
- `NSArray` и `NSMutableArray`. Неизменяемые и изменяемые массивы объектов. Массив — это упорядоченная коллекция элементов. Например, у вас может быть 10 строковых объектов, которые вы хотите сохранить в памяти. Для этого хорошо подойдет массив.
- `NSSet`, `NSMutableSet`, `NSOrderedSet`, `NSMutableOrderedSet`. Это типы множеств. Множества напоминают массивы тем, что могут содержать в себе наборы объектов, но в отличие от массива множество может включать в себя только уникальные объекты. Массив может содержать несколько экземпляров одного и того же объекта, а в множестве каждый объект может присутствовать только в одном экземпляре. Рекомендую вам четко усвоить разницу между массивами и множествами и использовать их правильно.
- `NSData` и `NSMutableData`. Неизменяемые и изменяемые контейнеры для любых данных. Такие типы данных очень вам пригодятся, если вы, например, хотите выполнить считывание содержимого файла в память.

Одни из рассмотренных нами типов данных являются примитивами, другие — классами. Вам придется просто запомнить, какие из них относятся к каждой из категорий. Например, тип данных NSInteger является примитивом, а NSString — классом. Поэтому из NSString можно создавать объекты. В языке Objective-C, как и в C и C++, существуют указатели. Указатель — это тип данных, в котором сохраняется адрес в памяти. По этому адресу уже хранятся фактические данные. Вы уже, наверное, знаете, что указатели на классы обозначаются символом астериска (*):

```
NSString *myString = @"Objective-C is great!";
```

Следовательно, если вы хотите присвоить строку переменной типа NSString на языке Objective-C, то вам понадобится просто сохранить данные в указатель типа NSString *. Но если вы собираетесь сохранить в переменной значение, представляющее собой число с плавающей точкой, то не сможете использовать указатель, так как тип данных, к которому относится эта переменная, не является классом:

```
/* Присваиваем переменной myFloat значение PI */
CGFloat myFloat = M_PI;
```

Если вам нужен указатель на эту переменную, соответствующую числу с плавающей точкой, то вы можете поступить так:

```
/* Присваиваем переменной myFloat значение PI */
CGFloat myFloat = M_PI;

/* Создаем переменную указателя, которая направлена на переменную myFloat */
CGFloat *pointerFloat = &myFloat;
```

Мы получаем данные от исходного числа с плавающей точкой путем простого разыменования (myFloat). Если получение значения происходит с применением указателя, то требуется использовать астериск (*pointerFloat). В некоторых ситуациях указатели могут быть полезны — например, при вызове функции, которая задает в качестве аргумента значение с плавающей точкой, а вы хотите получить новое значение после возврата функции.

Но вернемся к теме классов. Пожалуй, следует разобраться с ними немного подробнее, пока мы окончательно не запутались. Итак, приступим.

## Как создавать классы и правильно пользоваться ими

Класс — это структура данных, у которой могут быть методы, переменные экземпляра и свойства, а также многие другие черты. Но пока мы не будем углубляться в подробности и поговорим об основах работы с классами. Каждый класс должен следовать таким правилам.

- Класс должен наследовать от суперкласса. Из этого правила есть немногочисленные исключения. В частности, классы NSObject и NSProxy являются корневыми. У корневых классов не бывает суперкласса.
- Класс должен иметь имя, соответствующее Соглашению об именованиях методов в Cocoa.

- У класса должен быть файл интерфейса, в котором определяется интерфейс этого класса.
- У класса должна быть реализация, в которой вы прописываете все возможности, которые вы «обещали» предоставить согласно интерфейсу класса.

`NSObject` — это корневой класс, от которого наследуют практически все другие классы. В этом примере мы собираемся добавить класс под названием `Person` в проект, который был создан в подразделе «Создание и запуск вашего первого приложения для iOS» данного раздела. Далее мы добавим к этому классу два свойства, `firstName` и `lastName`, которые относятся к типу `NSString`. Выполните следующие шаги, чтобы создать класс `Person` и добавить его в ваш проект.

1. Откройте проект в Xcode и в меню File (Файл) выберите New ▸ File (Новый ▸ Файл).
2. Убедитесь, что слева, в разделе iOS, вы выбрали категорию Cocoa Touch. После этого выберите элемент Objective-C Class (Класс для Objective-C) и нажмите Next (Далее).
3. В разделе Class (Класс) введите имя `Person`.
4. В разделе Subclass of (Подкласс от) введите `NSObject`.

Когда справитесь с этим, нажмите кнопку Next (Далее). На данном этапе Xcode предложит вам сохранить этот файл. Просто сохраните новый класс в том каталоге, где находятся ваш проект и все его файлы. Это место выбирается по умолчанию. Затем нажмите кнопку Create (Создать) — и дело сделано.

После этого в ваш проект будут добавлены два новых файла: `Person.h` и `Person.m`. Первый файл — это интерфейс вашего класса `Person`, а второй — файл реализации этого класса. В Objective-C `.h`-файлы являются заголовочными. В таких файлах вы определяете интерфейс каждого файла. В `.m`-файле пишется сама реализация класса.

Теперь рассмотрим заголовочный файл нашего класса `Person` и определим для этого класса два свойства, имеющие тип `NSString`:

```
@interface Person : NSObject

@property (nonatomic, copy) NSString *firstName;
@property (nonatomic, copy) NSString *lastName;

@end
```

Как и переменные, свойства определяются в особом формате в следующем порядке.

1. Определение свойства должно начинаться с ключевого слова `@property`.
2. Затем следует указать квалификаторы свойства. Неатомарные (`nonatomic`) свойства не являются потокобезопасными. О безопасности потоков мы поговорим в главе 14. Вы можете указать и другие квалификаторы свойств: `assign`, `copy`, `weak`, `strong` или `unsafe_unretained`. Чуть позже мы подробнее поговорим и о них.
3. Затем укажите тип данных для свойства, например `NSInteger` или `NSString`.
4. Наконец, не забудьте задать имя для свойства. Имена свойств должны соответствовать рекомендациям Apple.

Как было указано ранее, свойства могут иметь различные квалификаторы. Вот важнейшие квалификаторы, в которых вы должны разбираться.

- `strong` — свойства этого типа будут сохраняться во время исполнения. Они могут быть только экземплярами классов. Иными словами, вы не можете сохранить значение в свойстве типа `strong`, если значение является примитивом. Можно сохранять объекты, но не примитивы.
- `copy` — аналогичен `strong`, но при выполнении присваивания к свойствам этого типа среда времени исполнения будет делать копию объекта в правой части операции присваивания. Объект, находящийся в правой части этой операции, должен соответствовать протоколу `NSCopying` или `NSMutableCopying`.
- `assign` — значения объектов или примитивов, задаваемые в качестве значения свойства типа `assign`, не будут копироваться или сохраняться этим свойством. Для свойств примитивов этот квалификатор будет создавать адрес в памяти, в котором вы сможете поместить информацию примитива. В случае с объектами свойства такого типа будут просто указывать на объект в правой части равенства.
- `unsafe_unretained` — аналогичен квалификатору `assign`.
- `weak` — практически аналогичен квалификатору `assign`, но с одним большим отличием. При работе с объектами, когда объект, присвоенный свойству такого типа, высвобождается из памяти, среда времени исполнения будет автоматически устанавливать значение этого свойства в `nil`.

Итак, у нас есть класс `Person` с двумя свойствами, `firstName` и `lastName`. Вернемся к файлу реализации делегата нашего приложения (`AppDelegate.m`) и создадим объект типа `Person`:

```
#import "AppDelegate.h"
#import "Person.h"
@implementation AppDelegate

- (BOOL) application:(UIApplication *)application
  didFinishLaunchingWithOptions:(NSDictionary *)launchOptions{

    Person *person = [[Person alloc] init];

    person.firstName = @"Steve";
    person.lastName = @"Jobs";

    self.window = [[UIWindow alloc]
    initWithFrame:[[UIScreen mainScreen] bounds]];
    self.window.backgroundColor = [UIColor whiteColor];
    [self.window makeKeyAndVisible];
    return YES;
}
```

В этом примере мы выделяем и инициализируем наш экземпляр класса `Person`. Возможно, вы еще не понимаете, что это значит, но в подразделе «Добавление функционала к классам с помощью методов», приведенном далее, мы подробно об этом поговорим.

## Добавление нового функционала к классам с помощью методов

Методы — это строительные блоки, из которых состоят классы. Например, класс `Person` может иметь логические возможности — обозначим их как «ходить», «дышать», «есть» и «пить». Обычно такие функции инкапсулируются в методах.

Метод может принимать параметры. Параметры — это переменные, передаваемые вызывающей стороной при вызове метода и видимые только этому методу. Например, в упрощенном мире у нашего класса `Person` был бы метод `walk`. Но вы могли бы добавить к этому методу параметр или аргумент и назвать его `walkingSpeed`. Этому параметру вы бы присвоили тип `CGFloat`. Теперь, если другой программист вызовет этот метод в вашем классе, он может указать, с какой скоростью будет идти `Person`. Вы как автор класса напишете соответствующий код, который будет обрабатывать различные скорости ходьбы `Person`. Не переживайте, если у вас возникает ощущение «как-то много работы получается». Рассмотрим следующий пример. В нем я добавил метод в файл реализации того класса `Person`, который мы создали в подразделе «Как создавать классы и правильно пользоваться ими» данного раздела.

```
#import "Person.h"

@implementation Person

- (void) walkAtKilometersPerHour:(CGFloat)paramSpeedKilometersPerHour{
/* здесь пишем код для этого метода */
}

- (void) runAt10KilometersPerHour{
  /* Вызываем метод walk в нашем собственном классе и передаем значение 10 */
  [self walkAtKilometersPerHour:10.0f];
}
@end
```

Типичный метод в языке Objective-C имеет следующие качества.

1. Префикс указывает компилятору, является ли данный код методом экземпляра (–) или методом класса (+). К методу экземпляра можно обратиться лишь после того, как программист выделит и инициализирует экземпляр вашего класса. Получить доступ к методу класса можно, вызвав его непосредственно из этого класса. Не волнуйтесь, если на первый взгляд это кажется сложным. В этой книге мы рассмотрим многочисленные примеры методов, пока просто следите за ходом рассказа.
2. Тип данных для метода, если метод возвращает какое-либо значение. В примере мы указали тип данных `void`. Так мы сообщаем компилятору, что не собираемся возвращать от метода какое-либо значение.
3. Первая часть имени метода, за которой идет первый параметр. Метод может и не иметь параметров. Методы, не принимающие параметров, довольно широко распространены.
4. Список последующих параметров, идущих за первым.

Рассмотрим пример метода с двумя параметрами:

```
- (void) singSong:(NSData *)paramSongData loudly:(BOOL)paramLoudly{
  /* Параметры, к которым мы можем обратиться здесь, в этом методе, таковы:

  paramSongData (для доступа к информации о песне)
  paramLoudly сообщает нам, должны мы петь песню громко или нет
  */
}
```

Важно учитывать, что каждый параметр каждого метода обладает *внешним* и *внутренним* именем. Внешнее имя входит в состав метода, а внутреннее имя — это фактическое название (или псевдоним) параметра, которое может использоваться в пределах реализации метода. В предыдущем примере внешнее имя первого параметра — `singSong`, а внутреннее — `paramSongData`. Внешнее имя второго параметра — `loudly`, а внутреннее — `paramLoudly`. Имя метода и внешние имена его параметров вместе образуют сущность, которая называется *селектором* метода. В данном случае селектор упомянутого метода будет иметь вид `singSong:loudly:`. Как будет объяснено далее в этой книге, селектор является идентификатором каждого метода в среде времени исполнения. Никакие два метода в рамках одного и того же класса не могут иметь одинаковые селекторы.

В нашем примере мы определили в файле реализации класса `Person` (`Person.m`) три метода:

- `walkAtKilometersPerHour:`;
- `runAt10KilometersPerHour`;
- `singSong:loudly:`.

Если бы мы хотели использовать любой из этих методов из какой-нибудь сущности, находящейся вне класса, например из делегата приложения, то должны были бы предоставить эти методы в нашем файле интерфейса (`Person.h`):

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

@interface Person : NSObject

@property (nonatomic, copy) NSString *firstName;
@property (nonatomic, copy) NSString *lastName;

- (void) walkAtKilometersPerHour:(CGFloat)paramSpeedKilometersPerHour;
- (void) runAt10KilometersPerHour;

/* Не предоставляем метод singSong:loudly: для доступа извне.
Этот метод является внутренним для нашего класса. Зачем же нам открывать к нему
доступ? */

@end
```

Имея такой файл интерфейса, программист может вызывать методы `walkAtKilometersPerHour:` и `runAt10KilometersPerHour` извне класса `Person`. А метод `singSong:loudly:` так вызывать нельзя, поскольку он не предоставлен в файле интерфейса.

Итак, продолжим: попробуем вызвать все три этих метода из делегата нашего приложения и посмотрим, что получится:

```
- (BOOL) application:(UIApplication *)application
  didFinishLaunchingWithOptions:(NSDictionary *)launchOptions{

    Person *person = [[Person alloc] init];

    [person walkAtKilometersPerHour:3.0f];
    [person runAt10KilometersPerHour];

    /* Если раскомментировать следующую строку кода, то компилятор выдаст
    вам ошибку и сообщит, что такого метода в классе Person не существует */
    //[person singSong:nil loudly:YES];

    self.window = [[UIWindow alloc]
    initWithFrame:[[UIScreen mainScreen] bounds]];
    self.window.backgroundColor = [UIColor whiteColor];
    [self.window makeKeyAndVisible];
    return YES;
}
```

Итак, теперь мы умеем определять и вызывать методы экземпляров. А что насчет методов классов? Сначала разберемся, что такое методы классов и чем они отличаются от методов экземпляров.

Метод экземпляра — это метод, относящийся к экземпляру класса. Например, в нашем случае вы можете создать экземпляр класса `Person` дважды и получить в гипотетической игре, которую разрабатываете, двух разных персонажей. Один персонаж будет ходить со скоростью 3 км/ч, другой — 2 км/ч.

Пусть вы и написали код для метода экземпляра `walk` всего один раз, но когда во время исполнения создаются два экземпляра класса `Person`, поступающие от них вызовы методов экземпляра маршрутизируются к соответствующему экземпляру класса (тому, который выполнил вызов).

Напротив, методы класса работают только с самим классом. Например, в вашей игре есть экземпляры класса `Light`, отвечающего за подсвечивание сцен в вашей игре. У этого класса может быть метод `dimAllLights`. Вызвав этот метод, программист погасит в игре все источники света независимо от того, где они находятся. Рассмотрим пример метода класса, применяемого с нашим классом `Person`:

```
#import "Person.h"

@implementation Person

+ (CGFloat) maximumHeightInCentimeters{
  return 250.0f;
}

+ (CGFloat) minimumHeightInCentimeters{
  return 40.0f;
```

```
}

@end
```

Метод `maximumHeightInCentimeters` — это метод класса, возвращающий гипотетический максимальный рост любого персонажа в сантиметрах. Метод класса `minimumHeightInCentimeters` возвращает минимальный рост *любого* персонажа. Вот как мы предоставим оба этих метода в файле интерфейса нашего класса:

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

@interface Person : NSObject

@property (nonatomic, copy) NSString *firstName;
@property (nonatomic, copy) NSString *lastName;
@property (nonatomic, assign) CGFloat currentHeight;

+ (CGFloat) maximumHeightInCentimeters;
+ (CGFloat) minimumHeightInCentimeters;

@end
```

Мы добавили к нашему классу Person еще одно свойство, принимающее значения с плавающей точкой. Оно называется currentHeight. С его помощью экземпляры этого класса могут хранить информацию о своей высоте в памяти (для справки) — точно так же, как имя и фамилию.

А в делегате нашего приложения мы продолжим работать с методами вот так:

```
- (BOOL) application:(UIApplication *)application
  didFinishLaunchingWithOptions:(NSDictionary *)launchOptions{

    Person *steveJobs = [[Person alloc] init];
    steveJobs.firstName = @"Steve";
    steveJobs.lastName = @"Jobs";
    steveJobs.currentHeight = 175.0f; /* Сантиметры */

    if (steveJobs.currentHeight >= [Person minimumHeightInCentimeters] &&
      steveJobs.currentHeight <= [Person maximumHeightInCentimeters]){
      /* Высота этого персонажа находится в пределах допустимого */
    } else {
      /* Высота этого персонажа находится вне пределов допустимого */
    }

    self.window = [[UIWindow alloc]
    initWithFrame:[[UIScreen mainScreen] bounds]];
    self.window.backgroundColor = [UIColor whiteColor];
    [self.window makeKeyAndVisible];
    return YES;
}
```

## Соблюдение требований, предъявляемых другими классами, с помощью протоколов

В языке Objective-C существует концепция под названием *«протокол»*. Протоколы встречаются и во многих других языках, но называются везде по-разному; например, в Java аналогичная сущность называется «интерфейс». Как понятно из названия, протокол — это набор правил, которым класс должен соответствовать, чтобы его можно было использовать тем или иным образом. Если класс выполняет правила определенного протокола, то принято говорить, что он *соответствует* этому протоколу. Протоколы отличаются от самих классов тем, что не имеют реализации. Это просто правила. Например, у любой машины есть колеса, дверцы и цвет кузова, а также многие другие свойства. Определим эти свойства в протоколе `Car`. Просто выполните следующие шаги, чтобы создать заголовочный файл, который может содержать наш протокол `Car`.

1. Откройте ваш проект в Xcode и в меню File (Файл) выберите New ▸ File (Новый ▸ Файл).
2. Убедитесь, что слева, в разделе iOS, вы выбрали категорию Cocoa Touch. После этого выберите элемент Objective-C Protocol (Протокол для Objective-C) и нажмите Next (Далее).
3. В разделе Class (Класс) введите имя `Car`, затем нажмите кнопку Next (Далее).
4. Далее система предложит вам сохранить ваш протокол на диске. Просто выберите для этого место (как правило, в каталоге с вашим проектом) и нажмите кнопку Create (Создать).

   После этого Xcode создаст для вас файл `Car.h` с таким содержимым:

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

@protocol Car <NSObject>
@end
```

Продолжим и определим свойства для протокола `Car`, как мы обсуждали ранее в этом разделе:

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

@protocol Car <NSObject>

@property (nonatomic, copy) NSArray *wheels;
@property (nonatomic, strong) UIColor *bodyColor;
@property (nonatomic, copy) NSArray *doors;

@end
```

Теперь, когда наш протокол определен, создадим класс, обозначающий автомобиль, — например, `Jaguar`, — а потом обеспечим соответствие этого класса протоколу. Просто выполните все шаги, перечисленные в подразделе «Как создавать классы и правильно пользоваться ими» данного раздела, после чего обеспечьте его соответствие протоколу `Car` следующим образом:

```
#import <Foundation/Foundation.h>
#import "Car.h"

@interface Jaguar : NSObject <Car>
@
end
```

Если вы попробуете собрать ваш проект на данном этапе, то компилятор выдаст вам несколько предупреждений, например такое:

```
Auto property synthesis will not synthesize property declared in a protocol
```

Это означает, что ваш класс `Jaguar` пытается соответствовать протоколу `Car`, но на самом деле не реализует всех требуемых свойств и/или методов, описанных в этом протоколе. Теперь вы уже знаете, что в протоколе могут содержаться необходимые и факультативные (опциональные) элементы, которые вы помечаете ключевыми словами `@optional` или `@required`. По умолчанию действует квалификатор `@required`, и поскольку мы явно не указываем квалификатор для этого протокола, компилятор неявно выбирает `@required` за нас. Следовательно, класс `Jaguar` теперь *обязан* реализовывать все аспекты, требуемые протоколом `Car`, вот так:

```
#import <Foundation/Foundation.h>
#import "Car.h"

@interface Jaguar : NSObject <Car>

@property (nonatomic, copy) NSArray *wheels;
@property (nonatomic, strong) UIColor *bodyColor;
@property (nonatomic, copy) NSArray *doors;

@end
```

Отлично. Теперь мы понимаем основы работы с протоколами, то, как они работают и как их определить. Далее в этой книге мы подробнее поговорим о протоколах, а на данный момент вы получили довольно полное представление о них.

## Хранение элементов в коллекциях и получение элементов из коллекций

Коллекции — это такие объекты, в экземплярах которых могут храниться другие объекты. Одна из самых распространенных разновидностей коллекций — это массив, который инстанцирует `NSArray` или `NSMutableArray`. В массиве можно хранить любой объект, причем массив может содержать несколько экземпляров одного и того же объекта. В следующем примере мы создаем массив из трех строк:

```
NSArray *stringsArray = @[
                          @"String 1",
                          @"String 2",
```

```
                        @"String 3"
                        ];

__unused NSString *firstString = stringsArray[0];
__unused NSString *secondString = stringsArray[1];
__unused NSString *thirdString = stringsArray[2];
```

Макрос __unused приказывает компилятору «не жаловаться», когда переменная — в нашем случае переменная firstString — объявлена, но ни разу не использовалась. По умолчанию в такой ситуации компилятор выдает в консоль предупреждение, сообщающее, что переменная не используется. В нашем кратком примере мы объявили переменные, но не задействовали их. Поэтому, если добавить вышеупомянутый макрос в начале объявления переменной, это вполне устроит и нас, и компилятор.

Изменяемый массив — это такой массив, в который можно вносить изменения уже после того, как он был создан. Как мы видели ранее, неизменяемый массив не может быть дополнен новой информацией уже после создания. Вот пример неизменяемого массива:

```
NSString *string1 = @"String 1";
NSString *string2 = @"String 2";
NSString *string3 = @"String 3";

NSArray *immutableArray = @[string1, string2, string3];

NSMutableArray *mutableArray = [[NSMutableArray alloc]
initWithArray:immutableArray];

[mutableArray exchangeObjectAtIndex:0 withObjectAtIndex:1];
[mutableArray removeObjectAtIndex:1];
[mutableArray setObject:string1 atIndexedSubscript:0];
NSLog(@"Immutable array = %@", immutableArray);
NSLog(@"Mutable Array = %@", mutableArray);
```

Вывод этой программы таков:

```
Immutable array = (
    "String 1",
    "String 2",
    "String 3"
)
Mutable Array = (
    "String 1",
    "String 3"
)
```

Еще одна распространенная коллекция, которая часто встречается в программах для iOS, — это *словарь*. Словари похожи на массивы, но каждому объекту в словаре присваивается ключ, и по этому ключу вы можете позже получить интересующий вас объект. Рассмотрим пример:

```
NSDictionary *personInformation =
@{
```

```
  @"firstName" : @"Mark",
  @"lastName" : @"Tremonti",
  @"age" : @30,
  @"sex" : @"Male"
};

NSString *firstName = personInformation[@"firstName"];
NSString *lastName = personInformation[@"lastName"];
NSNumber *age = personInformation[@"age"];
NSString *sex = personInformation[@"sex"];

NSLog(@"Full name = %@ %@", firstName, lastName);
NSLog(@"Age = %@, Sex = %@", age, sex);
```

А вот и вывод этой программы:

```
Full name = Mark Tremonti
Age = 30, Sex = Male
```

Можно также использовать изменяемые словари, которые довольно сильно похожи на изменяемые массивы. Содержимое изменяемого словаря можно изменить после того, как словарь инстанцирован. Пример:

```
NSDictionary *personInformation =
@{
  @"firstName" : @"Mark",
  @"lastName" : @"Tremonti",
  @"age" : @30,
  @"sex" : @"Male"
};

NSMutableDictionary *mutablePersonInformation =
  [[NSMutableDictionary alloc] initWithDictionary:personInformation];
mutablePersonInformation[@"age"] = @32;

NSLog(@"Information = %@", mutablePersonInformation);
```

Вывод этой программы таков:

```
Information = {
    age = 32;
    firstName = Mark;
    lastName = Tremonti;
    sex = Male;
}
```

Еще можно работать с множествами. Множества похожи на массивы, но любой объект, входящий в состав множества, должен встречаться в нем только один раз. Иными словами, в одном множестве не может быть двух экземпляров одного и того же объекта. Пример множества:

```
NSSet *shoppingList = [[NSSet alloc] initWithObjects:
                       @"Milk",
                       @"Bananas",
```

```
                        @"Bread",
                        @"Milk", nil];

NSLog(@"Shopping list = %@", shoppingList);
```

Запустив эту программу, вы получите следующий вывод:

```
Shopping list = {(
    Milk,
    Bananas,
    Bread
)}
```

Обратите внимание: элемент `Milk` упомянут в программе дважды, а в множество добавлен всего один раз. Эта черта множеств — настоящее волшебство. Изменяемые множества можно использовать и вот так:

```
NSSet *shoppingList = [[NSSet alloc] initWithObjects:
                        @"Milk",
                        @"Bananas",
                        @"Bread",
                        @"Milk", nil];

NSMutableSet *mutableList = [NSMutableSet setWithSet:shoppingList];

[mutableList addObject:@"Yogurt"];
[mutableList removeObject:@"Bread"];
NSLog(@"Original list = %@", shoppingList);
NSLog(@"Mutable list = %@", mutableList);
```

А вывод будет таким:

```
Original list = {(
    Milk,
    Bananas,
    Bread
)}
Mutable list = {(
    Milk,
    Bananas,
    Yogurt
)}
```

Обсуждая множества и коллекции, следует упомянуть еще два важных класса, о которых вам необходимо знать:

- `NSOrderedSet` — неизменяемое множество, учитывающее, в каком порядке в него добавлялись объекты;
- `NSMutableOrderedSet` — изменяемый вариант вышеупомянутого изменяемого множества.

По умолчанию множества не учитывают, в каком порядке объекты в них добавлялись. Рассмотрим пример:

```
NSSet *setOfNumbers = [NSSet setWithArray:@[@3, @4, @1, @5, @10]];
NSLog(@"Set of numbers = %@", setOfNumbers);
```

Запустив эту программу, получим на экране следующий вывод:

```
Set of numbers = {(
    5,
    10,
    3,
    4,
    1
)}
```

Но на самом деле мы наполняли множество элементами в другом порядке. Если вы хотите сохранить правильный порядок, просто воспользуйтесь классом `NSOrderedSet`:

```
NSOrderedSet *setOfNumbers = [NSOrderedSet orderedSetWithArray
                              :@[@3, @4, @1, @5, @10]];

NSLog(@"Ordered set of numbers = %@", setOfNumbers);
```

Разумеется, вы можете воспользоваться и изменяемой версией упорядоченного множества:

```
NSMutableOrderedSet *setOfNumbers =
  [NSMutableOrderedSet orderedSetWithArray:@[@3, @4, @1, @5, @10]];

[setOfNumbers removeObject:@5];
[setOfNumbers addObject:@0];
[setOfNumbers exchangeObjectAtIndex:1 withObjectAtIndex:2];

NSLog(@"Set of numbers = %@", setOfNumbers);
```

А вот и результаты:

```
Set of numbers = {(
    3,
    1,
    4,
    10,
    0
)}
```

Прежде чем завершить разговор о множествах, упомяну еще об одном удобном классе, который может вам пригодиться. Класс `NSCountedSet` может несколько раз содержать уникальный экземпляр объекта. Правда, в нем эта задача решается иначе, нежели в массивах. В массиве может несколько раз присутствовать один и тот же объект. А в рассматриваемом здесь «подсчитываемом множестве» каждый объект появляется в множестве как будто заново, но множество ведет подсчет того, сколько раз объект был добавлен в множество, и снижает значение этого счетчика на единицу, как только вы удалите из этого множества экземпляр данного объекта. Вот пример:

```
NSCountedSet *setOfNumbers = [NSCountedSet setWithObjects:
                              @10, @20, @10, @10, @30, nil];

[setOfNumbers addObject:@20];
[setOfNumbers removeObject:@10];

NSLog(@"Count for object @10 = %lu",
    (unsigned long)[setOfNumbers countForObject:@10]);

NSLog(@"Count for object @20 = %lu",
    (unsigned long)[setOfNumbers countForObject:@20]);
```

Вывод программы:

```
Count for object @10 = 2
Count for object @20 = 2
```

Класс NSCountedSet является изменяемым, хотя из его названия это и не следует.

## Обеспечение поддержки подписывания объектов в ваших классах

Традиционно при необходимости доступа к объектам, содержащимся в коллекциях — например, массивах и словарях, — программисту требовалось получить доступ к методу в словаре или массиве, чтобы получить или установить желаемый объект. Например, создавая изменяемый словарь, мы добавляем в него два ключа и значения, получая эти значения обратно:

```
NSString *const kFirstNameKey = @"firstName";
NSString *const kLastNameKey = @"lastName";

NSMutableDictionary *dictionary = [[NSMutableDictionary alloc] init];
[dictionary setValue:@"Tim" forKey:kFirstNameKey];
[dictionary setValue:@"Cook" forKey:kLastNameKey];

__unused NSString *firstName = [dictionary valueForKey:kFirstNameKey];
__unused NSString *lastName = [dictionary valueForKey:kLastNameKey];
```

Но с развитием компилятора LLVM этот код можно сократить, придав ему следующий вид:

```
NSString *const kFirstNameKey = @"firstName";
NSString *const kLastNameKey = @"lastName";

NSDictionary *dictionary = @{
                            kFirstNameKey : @"Tim",
                            kLastNameKey : @"Cook",
```

```
                                    };

__unused NSString *firstName = dictionary[kFirstNameKey];
__unused NSString *lastName = dictionary[kLastNameKey];
```

Как видите, мы инициализируем словарь, давая ключи в фигурных скобках. Точно так же можно поступать и с массивами. Вот как мы обычно создаем и используем массивы:

```
NSArray *array = [[NSArray alloc] initWithObjects:@"Tim", @"Cook", nil];
__unused NSString *firstItem = [array objectAtIndex:0];
__unused NSString *secondObject = [array objectAtIndex:1];
```

А теперь, имея возможность подписывать объекты, мы можем сократить этот код следующим образом:

```
NSArray *array = @[@"Tim", @"Cook"];
__unused NSString *firstItem = array[0];
__unused NSString *secondObject = array[0];
```

Компилятор LLVM не останавливается и на этом. Вы можете также добавлять подписывание и к собственным классам. Существует два типа подписывания:

- *подписывание по ключу* — действуя таким образом, вы можете задавать внутри объекта значение для того или иного ключа точно так же, как вы делали бы это в словаре. Указывая ключ, вы также можете получать доступ к значениям внутри объекта и считывать их;
- *подписывание по индексу* — как и при работе с массивами, вы можете устанавливать/получать значения внутри объекта, предоставив для этого объекта индекс. Это целесообразно делать в массивоподобных классах, где элементы естественным образом располагаются в порядке, удобном для индексирования.

Сначала рассмотрим пример подписывания по ключу. Для этого создадим класс под названием `Person`, имеющий свойства `firstName` и `lastName`. Далее мы позволим программисту менять значения этих свойств (имя и фамилию), просто предоставив ключи для этих свойств.

Вам может понадобиться добавить к классу подобный механизм подписывания по ключу, например, по такой причине: имена ваших свойств могут изменяться и вы хотите предоставить программисту возможность устанавливать значения таких свойств, не учитывая, будут ли имена этих свойств впоследствии изменяться. В противном случае программисту лучше будет использовать свойства напрямую. Другая причина реализации подписывания по ключу — стремление скрыть точную реализацию/объявление ваших свойств от программиста и закрыть программисту прямой доступ к этим свойствам.

Чтобы обеспечить поддержку подписывания по ключу в ваших собственных классах, вы должны реализовать в вашем классе два следующих метода *и* записать сигнатуры методов в файле заголовков этого класса. В противном случае компилятор не узнает, что в вашем классе поддерживается подписывание по ключу.

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

/* Мы будем использовать их как ключи для наших свойств firstName
```

```
  и lastName, так что если имена наших свойств firstName и lastName
  в будущем изменятся в реализации, нам не придется ничего переделывать
  и наш класс останется работоспособным, поскольку мы сможем просто
  изменить значения этих констант в нашем файле реализации */
extern NSString *const kFirstNameKey;
extern NSString *const kLastNameKey;

@interface Person : NSObject

@property (nonatomic, copy) NSString *firstName;
@property (nonatomic, copy) NSString *lastName;

- (id) objectForKeyedSubscript:(id<NSCopying>)paramKey;
- (void) setObject:(id)paramObject forKeyedSubscript:(id<NSCopying>)paramKey;

@end
```

Метод `objectForKeyedSubscript:` будет вызываться в вашем классе всякий раз, когда программист предоставит ключ и захочет прочитать в вашем классе значение, соответствующее данному ключу. Очевидно, тот параметр, который будет вам передан, будет представлять собой ключ, по которому программист хочет считать интересующее его значение. Дополнительно к этому методу мы будем вызывать в нашем классе метод `setObject:forKeyedSubscript:` всякий раз, когда программист захочет задать значение для конкретного ключа. Итак, в данной реализации мы хотим проверить, ассоциированы ли заданные ключи с именами и фамилиями. Если это так, то собираемся установить/получить в нашем классе значения имени и фамилии:

```
#import "Person.h"

NSString *const kFirstNameKey = @"firstName";
NSString *const kLastNameKey = @"lastName";

@implementation Person

- (id) objectForKeyedSubscript:(id<NSCopying>)paramKey{

  NSObject<NSCopying> *keyAsObject = (NSObject<NSCopying> *)paramKey;
  if ([keyAsObject isKindOfClass:[NSString class]]){
    NSString *keyAsString = (NSString *)keyAsObject;
    if ([keyAsString isEqualToString:kFirstNameKey] ||
      [keyAsString isEqualToString:kLastNameKey]){
      return [self valueForKey:keyAsString];
    }
  }

  return nil;
}

- (void) setObject:(id)paramObject forKeyedSubscript:(id<NSCopying>)paramKey{
  NSObject<NSCopying> *keyAsObject = (NSObject<NSCopying> *)paramKey;
  if ([keyAsObject isKindOfClass:[NSString class]]){
```

```
    NSString *keyAsString = (NSString *)keyAsObject;
    if ([keyAsString isEqualToString:kFirstNameKey] ||
        [keyAsString isEqualToString:kLastNameKey]){
        [self setValue:paramObject forKey:keyAsString];
    }
  }
}

@end
```

Итак, в этом коде мы получаем ключ в методе `objectForKeyedSubscript:`, а в ответ должны вернуть объект, который ассоциирован в нашем экземпляре с этим ключом. Ключ, который получаем, — это объект, соответствующий протоколу `NSCopying`. Это означает, что при желании мы можем сделать копию такого объекта. Рассчитываем на то, что ключ будет представлять собой строку, чтобы мы могли сравнить его с готовыми ключами, которые были заранее объявлены в начале класса. В случае совпадения зададим значение данного свойства в этом классе. После этого воспользуемся методом `valueForKey:`, относящимся к объекту `NSObject`, чтобы вернуть значение, ассоциированное с заданным ключом. Но, разумеется, прежде, чем так поступить, мы должны гарантировать, что данный ключ — один из тех, которые мы ожидаем. В методе `setObject:forKeyedSubscript:` мы делаем совершенно противоположное — устанавливаем значения для заданного ключа, а не возвращаем их.

Теперь в любой части вашего приложения вы можете инстанцировать объект типа `Person` и использовать заранее определенные ключи `kFirstNameKey` и `kLastNameKey`, чтобы изменить значения свойств `firstName` и `lastName`, вот так:

```
Person *person = [Person new];
person[kFirstNameKey] = @"Tim";
person[kLastNameKey] = @"Cook";
__unused NSString *firstName = person[kFirstNameKey];
__unused NSString *lastName = person[kLastNameKey];
```

Этот код позволяет достичь точно того же результата, что и при более лобовом подходе, когда мы устанавливаем свойства класса:

```
Person *person = [Person new];
person.firstName = @"Tim";
person.lastName = @"Cook";
__unused NSString *firstName = person.firstName;
__unused NSString *lastName = person.lastName;
```

Вы также можете поддерживать и подписывание по индексу — точно как при работе с массивами. Как было указано ранее, это полезно делать, чтобы обеспечивать программисту доступ к объектам, выстраиваемым в классе в некоем естественном порядке. Но, кроме массивов, существует не так уж много структур данных, где целесообразно упорядочивать и нумеровать элементы, чего не скажешь о подписывании по ключу, которое применяется в самых разных структурах данных. Поэтому пример, которым иллюстрируется подписывание по индексу, немного надуман. В предыдущем примере у нас существовал класс `Person` с именем и фамилией. Теперь мы хотим предоставить программистам возможность считывать имя, указывая

индекс 0, а фамилию — указывая индекс 1. Все, что требуется сделать для этого, — объявить методы `objectAtIndexedSubscript:` и `setObject:atIndexedSubscript:` в заголовочном файле класса, а затем написать реализацию. Вот как мы объявляем два этих метода в заголовочном файле класса `Person`:

```
- (id) objectAtIndexedSubscript:(NSUInteger)paramIndex;
- (void) setObject:(id)paramObject atIndexedSubscript:(NSUInteger)paramIndex;
```

Реализация также довольно проста. Мы берем индекс и оперируем им так, как это требуется в нашем классе. Ранее мы решили, что у имени должен быть индекс 0, а у фамилии — индекс 1. Итак, получаем индекс 0 для задания значения, присваиваем значение имени первому входящему объекту и т. д.:

```
- (id) objectAtIndexedSubscript:(NSUInteger)paramIndex{

  switch (paramIndex){
    case 0:{
      return self.firstName;
      break;
    }
    case 1:{
      return self.lastName;
      break;
    }
    default:{
      [NSException raise:@"Invalid index" format:nil];
    }
  }

  return nil;
}

- (void) setObject:(id)paramObject atIndexedSubscript:(NSUInteger)paramIndex{
  switch (paramIndex){
    case 0:{
      self.firstName = paramObject;
      break;
    }
    case 1:{
      self.lastName = paramObject;
      break;
    }
    default:{
      [NSException raise:@"Invalid index" format:nil];
    }
  }
}
```

Теперь можно протестировать весь написанный ранее код вот так:

```
Person *person = [Person new];
person[kFirstNameKey] = @"Tim";
```

```
person[kLastNameKey] = @"Cook";
NSString *firstNameByKey = person[kFirstNameKey];
NSString *lastNameByKey = person[kLastNameKey];

NSString *firstNameByIndex = person[0];
NSString *lastNameByIndex = person[1];

if ([firstNameByKey isEqualToString:firstNameByIndex] &&
    [lastNameByKey isEqualToString:lastNameByIndex]){
    NSLog(@"Success");
} else {
    NSLog(@"Something is not right");
}
```

Если вы правильно выполнили все шаги, описанные в этом разделе, то на консоли должно появиться значение `Success`.

# 1.1. Отображение предупреждений с помощью UIAlertView

## Постановка задачи

Вы хотите, чтобы у ваших пользователей отобразилось сообщение, которое будет оформлено как предупреждение (Alert). Такие сообщения можно применять, чтобы попросить пользователя подтвердить выбранное действие, запросить у него имя и пароль или просто предложить ввести какой-нибудь простой текст, который вы сможете использовать в своем приложении.

## Решение

Воспользуйтесь `UIAlertView`.

## Обсуждение

Если вы сами пользуетесь iOS, то вам определенно попадались виды-предупреждения. Пример такого вида показан на рис. 1.1.

Наилучший способ инициализации вида-предупреждения заключается, разумеется, в использовании его базового конструктора-инициализатора:

```
- (void) viewDidAppear:(BOOL)paramAnimated{

  [super viewDidAppear:paramAnimated];

  UIAlertView *alertView = [[UIAlertView alloc]
                            initWithTitle:@"Alert"
                            message:@"You've been delivered an alert"
                            delegate:nil
```

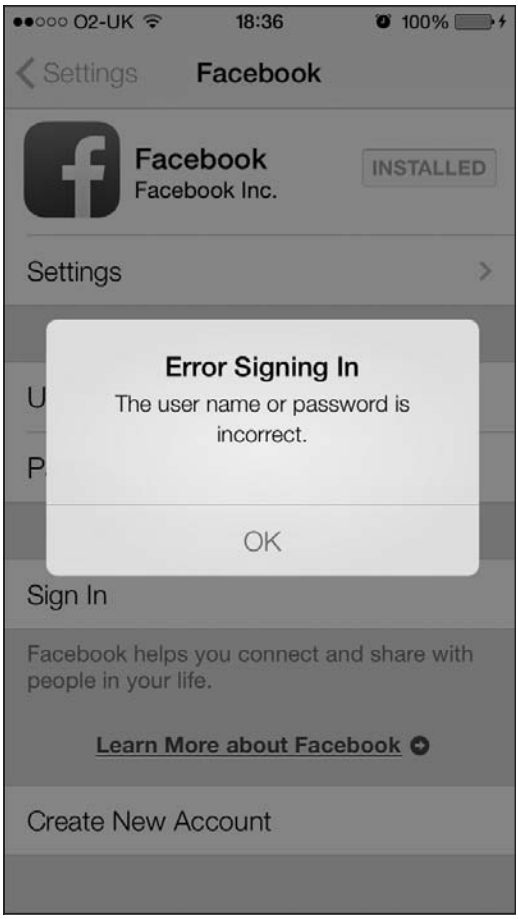


**Рис. 1.1.** Вид-предупреждение, сообщающий пользователю, что для работы требуется активное соединение с Интернетом

```
                          cancelButtonTitle:@"Cancel"
                          otherButtonTitles:@"OK", nil];
  [alertView show];
}
```

Когда этот вид-предупреждение отобразится у пользователя, он увидит экран, подобный показанному на рис. 1.2.

Чтобы показать пользователю вид-предупреждение, мы используем метод предупреждения `show`. Рассмотрим описания всех параметров, которые могут быть переданы базовому конструктору-инициализатору вида-предупреждения:

- `title` — строка, которую пользователь увидит в верхней части вида-предупрежения. На рис. 1.2 эта строка — `Title`;
- `message` — сообщение, которое отображается у пользователя. На рис. 1.2 для этого сообщения задано значение `Message`;

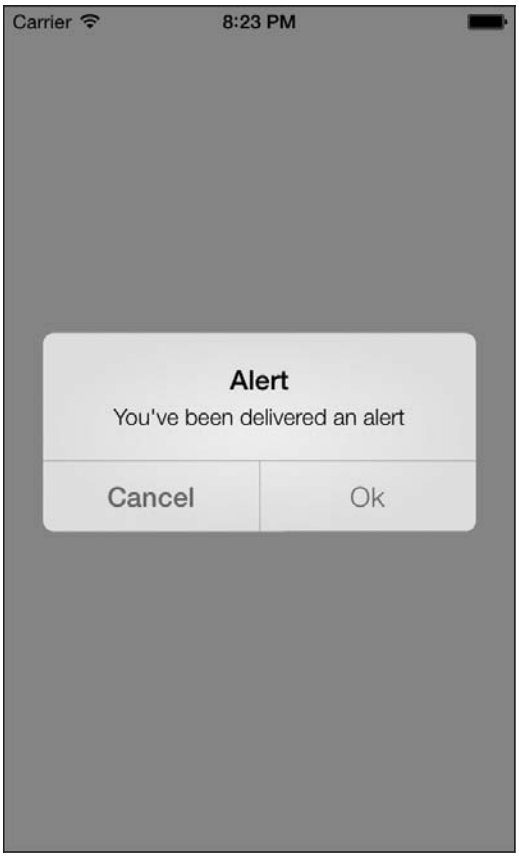


**Рис. 1.2.** Простой вид-предупреждение, отображаемый у пользователя

- `delegate` — опциональный объект-делегат, который мы передаем виду-предупреждению. Затем этот объект будет получать уведомление при каждом изменении состояния предупреждения, например, когда пользователь нажмет на экранную кнопку, изображенную в этом виде. Объект, передаваемый данному параметру, должен соответствовать протоколу `UIAlertViewDelegate`;
- `cancelButtonTitle` — строка, которая будет присваиваться кнопке отмены (Cancel Button) в виде-предупреждении. Если в виде-предупреждении есть кнопка отмены, то такой вид обычно побуждает пользователя к действию. Если пользователь не хочет совершать предложенное действие, то он нажимает кнопку отмены. Причем на этой кнопке не обязательно должна быть строка-надпись Cancel (Отменить). Надпись для этой кнопки определяете вы сами, и этот параметр опциональный — можно сделать диалоговое окно и без кнопки Отмена;
- `otherButtonTitles` — надписи на других кнопках, тех, которые вы хотите отобразить в виде-предупреждении. Разделяйте такие надписи запятыми. Нужно убедиться, что в конце списка названий стоит значение `nil`, называемое *сигнальной меткой*. Этот параметр не является обязательным.

Можно создать предупреждение вообще без кнопок. Но такое окно пользователь никак не сможет убрать с экрана. Создавая такой вид, вы как программист должны позаботиться о том, чтобы он убирался автоматически, например, через 3 секунды после того, как появится. Вид-предупреждение без кнопок, который не убирается автоматически, — это настоящее бедствие, с точки зрения пользователя. Ваше приложение не только получит низкие оценки на App Store за то, что вид-предупреждение блокирует пользовательский интерфейс. Велика вероятность, что вашу программу вообще удалят с рынка.

Виды-предупреждения можно оформлять с применением различных стилей. В классе `UIAlertView` есть свойство `alertViewStyle` типа `UIAlertViewStyle`:

```
typedef NS_ENUM(NSInteger, UIAlertViewStyle) {
  UIAlertViewStyleDefault = 0,
  UIAlertViewStyleSecureTextInput,
  UIAlertViewStylePlainTextInput,
  UIAlertViewStyleLoginAndPasswordInput
};
```

Вот что делает каждый из этих стилей:

- `UIAlertViewStyleDefault` — стандартный стиль вида-предупреждения, подобное оформление мы видели на рис. 1.2;
- `UIAlertViewStyleSecureTextInput` — при таком стиле в виде-предупреждении будет содержаться защищенное текстовое поле, которое станет скрывать от зрителя символы, вводимые пользователем. Такой вариант предупреждения вам подойдет, например, если вы запрашиваете у пользователя его учетные данные для дистанционного банковского обслуживания;
- `UIAlertViewStylePlainTextInput` — при таком стиле у пользователя будет отображаться незащищенное текстовое поле. Этот стиль отлично подходит для случаев, когда вы просите пользователя ввести несекретную последовательность символов, например номер его телефона;
- `UIAlertViewStyleLoginAndPasswordInput` — при таком стиле в виде-предупреждении будет два текстовых поля: незащищенное — для имени пользователя и защищенное — для пароля.

Если вам необходимо получать уведомление, когда пользователь начинает работать с видом-предупреждением, укажите объект-делегат для вашего предупреждения. Этот делегат должен подчиняться протоколу `UIAlertViewDelegate`. Самый важный метод, определяемый в этом протоколе, — `alertView:clickedButtonAtIndex:`, который вызывается сразу же, как только пользователь нажимает на одну из кнопок в виде-предупреждении. Индекс нажатой кнопки передается вам через параметр `clickedButtonAtIndex`.

В качестве примера отобразим предупреждение пользователю и спросим, хочет ли он перейти на сайт в браузере Safari после того, как нажмет ссылку на этот сайт, присутствующую в нашем пользовательском интерфейсе. В предупреждении будут отображаться две кнопки: Yes (Да) и No (Нет). В делегате вида-предупреждения мы увидим, какая кнопка была нажата, и предпримем соответствующие действия.

Сначала реализуем два очень простых метода, которые возвращают надпись на той или иной из двух кнопок:

```
- (NSString *) yesButtonTitle{
  return @"Yes";
}

- (NSString *) noButtonTitle{
  return @"No";
}
```

Теперь нужно убедиться, что контроллер нашего вида подчиняется протоколу `UIAlertViewDelegate`:

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

#import "ViewController.h"

@interface ViewController () <UIAlertViewDelegate>

@end

@implementation ViewController

...
```

Следующий шаг — создать и отобразить для пользователя окно с предупреждением:

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

  self.view.backgroundColor = [UIColor whiteColor];

  NSString *message = @"Are you sure you want to open this link in Safari?";
  UIAlertView *alertView = [[UIAlertView alloc]
                            initWithTitle:@"Open Link"
                            message:message
                            delegate:self
                            cancelButtonTitle:[self noButtonTitle]
                            otherButtonTitles:[self yesButtonTitle], nil];
  [alertView show];

}
```

Вид-предупреждение будет выглядеть примерно как на рис. 1.3.

Далее нужно узнать, какой вариант пользователь выбрал в нашем окне — No (Нет) или Yes (Да). Для этого потребуется реализовать метод `alertView:clickedButtonAtIndex:`, относящийся к делегату нашего вида-предупреждения:

```
- (void) alertView:(UIAlertView *)alertView
  clickedButtonAtIndex:(NSInteger)buttonIndex{

    NSString *buttonTitle = [alertView buttonTitleAtIndex:buttonIndex];

    if ([buttonTitle isEqualToString:[self yesButtonTitle]]){
```


**Рис. 1.3.** Вид-предупреждение с кнопками No (Нет) и Yes (Да)

```
    NSLog(@"User pressed the Yes button.");
  }
  else if ([buttonTitle isEqualToString:[self noButtonTitle]]){
    NSLog(@"User pressed the No button.");
  }

}
```

Стоит учитывать, что в больших проектах, когда несколько специалистов разрабатывают один и тот же исходный код, обычно удобнее сравнивать надписи с кнопок из вида-предупреждения с соответствующими строками, а не проверять, какая кнопка была нажата, ориентируясь на индекс этой кнопки. Чтобы решение с индексом работало, программисту придется найти код, в котором был сконструирован вид с предупреждением, и уже в этом коде посмотреть, у какой кнопки какой индекс. В рассмотренном же нами решении любой разработчик, даже не знающий, как именно был создан вид с предупреждением, может понять, какой оператор if что именно делает.

Как видите, мы пользуемся методом buttonTitleAtIndex: класса UIAlertView. Мы передаем этому методу индекс кнопки, отсчитываемый с нуля (кнопка находится в нашем виде), и получаем строку, которая представляет собой надпись на этой кнопке — если такая надпись вообще имеется. С помощью этого метода можно определить, какую кнопку нажал пользователь. Индекс этой кнопки будет передан нам как параметр buttonIndex метода alertView:clickedButtonAtIndex:. Если вас интересует надпись на этой кнопке, то нужно будет использовать метод buttonTitleAtIndex: класса UIAlertView. Все готово!

Кроме того, вид-предупреждение можно использовать и для текстового ввода, например, запрашивая у пользователя номер кредитной карточки или адрес. Для этого, как было указано ранее, нужно использовать стиль оформления предупреждения UIAlertViewStylePlainTextInput:

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

  UIAlertView *alertView = [[UIAlertView alloc]
                            initWithTitle:@"Credit Card Number"
                            message:@"Please enter your credit card number:"
                            delegate:self
                            cancelButtonTitle:@"Cancel"
                            otherButtonTitles:@"OK", nil];
  [alertView setAlertViewStyle:UIAlertViewStylePlainTextInput];
  /* Отобразить для этого текстового поля числовую клавиатуру. */
  UITextField *textField = [alertView textFieldAtIndex:0];
  textField.keyboardType = UIKeyboardTypeNumberPad;
  [alertView show];

}
```

Если сейчас запустить приложение в эмуляторе, то мы увидим такое изображение, как на рис. 1.4.

В этом коде мы изменяем стиль оформления вида на UIAlertViewStylePlainTextInput, а также делаем еще кое-что. Мы получили ссылку на первое и единственное текстовое поле, которое, как мы знаем, будет присутствовать в виде-предупреждении. Ссылку на текстовое поле применили для того, чтобы изменить тип клавиатуры, связанной с текстовым полем. Подробнее о текстовых полях поговорим в разделе 1.19.

Кроме обычного текста мы можем попросить пользователя набрать и защищенный текст. Как правило, защищается такой текст, который является для пользователя конфиденциальным, например пароль (рис. 1.5). Рассмотрим пример:

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

  UIAlertView *alertView = [[UIAlertView alloc]
                            initWithTitle:@"Password"
                           message:@"Please enter your password:"
                            delegate:self
                            cancelButtonTitle:@"Cancel"
```

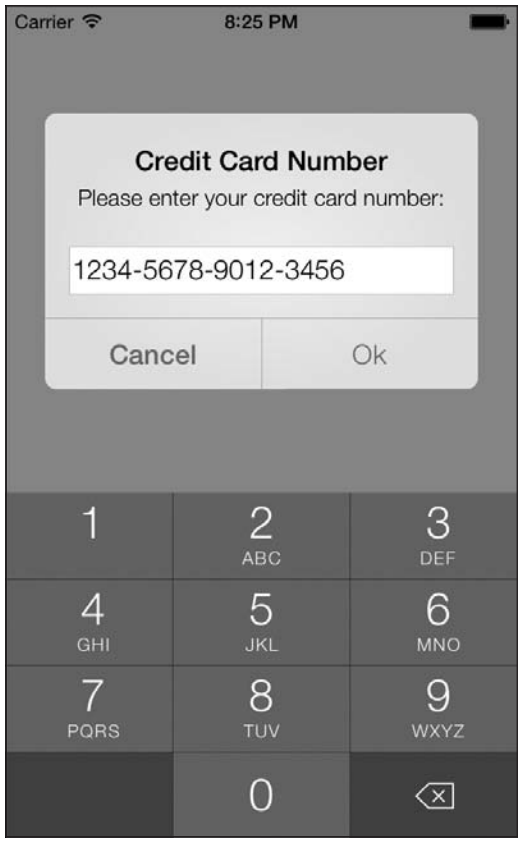


**Рис. 1.4.** Вид-предупреждение для ввода обычным текстом

```
                          otherButtonTitles:@"OK", nil];

  [alertView setAlertViewStyle:UIAlertViewStyleSecureTextInput];
  [alertView show];

}
```

Стиль `UIAlertViewStyleSecureTextInput` очень напоминает `UIAlertViewStyle-PlainTextInput`, за исключением того, что вместо символов текста мы подставляем какие-то нейтральные символы.

Следующий стиль довольно полезный. Он позволяет отобразить два текстовых поля: одно для имени пользователя, а другое — для пароля. Текст в первом поле открыт, а во втором — скрыт:

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

  UIAlertView *alertView = [[UIAlertView alloc]
                            initWithTitle:@"Password"
```

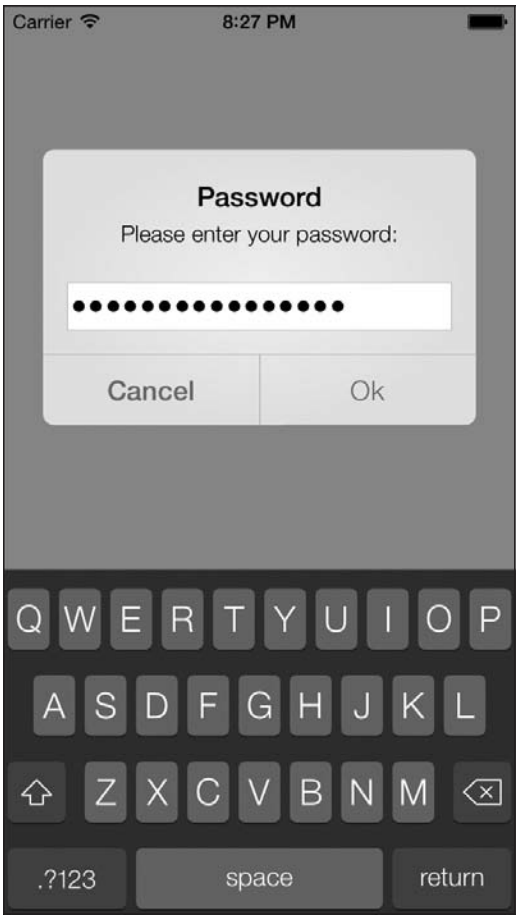


**Рис. 1.5.** Ввод защищенного текста в окно с предупреждением

```
                    message:@"Please enter your credentials:"
                    delegate:self
                    cancelButtonTitle:@"Cancel"
                    otherButtonTitles:@"OK", nil];

  [alertView setAlertViewStyle:UIAlertViewStyleLoginAndPasswordInput];
  [alertView show];

}
```

В результате увидим такое изображение, как на рис. 1.6.

## См. также

Раздел 1.19.

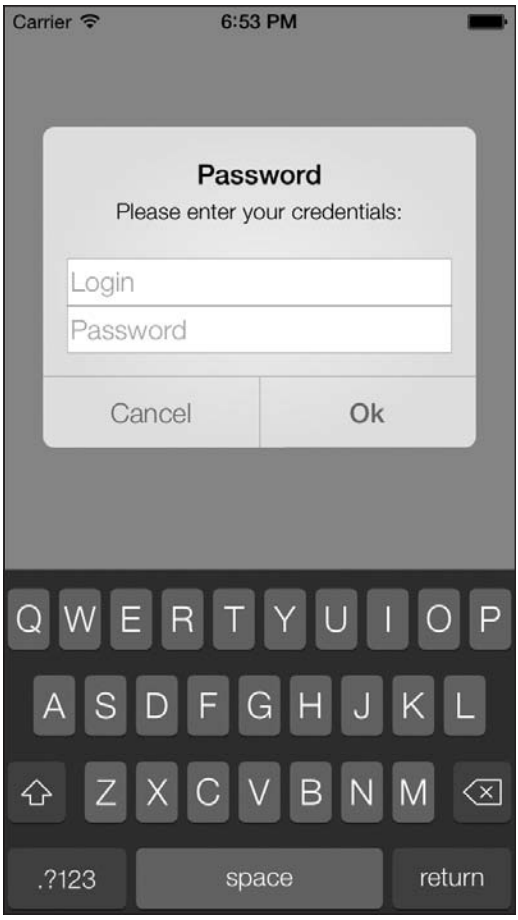


**Рис. 1.6.** Стиль, позволяющий вводить в вид-предупреждение имя пользователя и пароль

# 1.2. Создание и использование переключателей с помощью UISwitch

## Постановка задачи

Вы хотите дать пользователям возможность включать и отключать определенные функции.

## Решение

Воспользуйтесь классом `UISwitch`.

## Обсуждение

Класс `UISwitch` предоставляет инструмент управления ON/OFF (Вкл./Выкл.), как на рис. 1.7. Этот инструмент используется для работы с автоматической капитализацией, автоматическим исправлением орфографических ошибок и т. д.

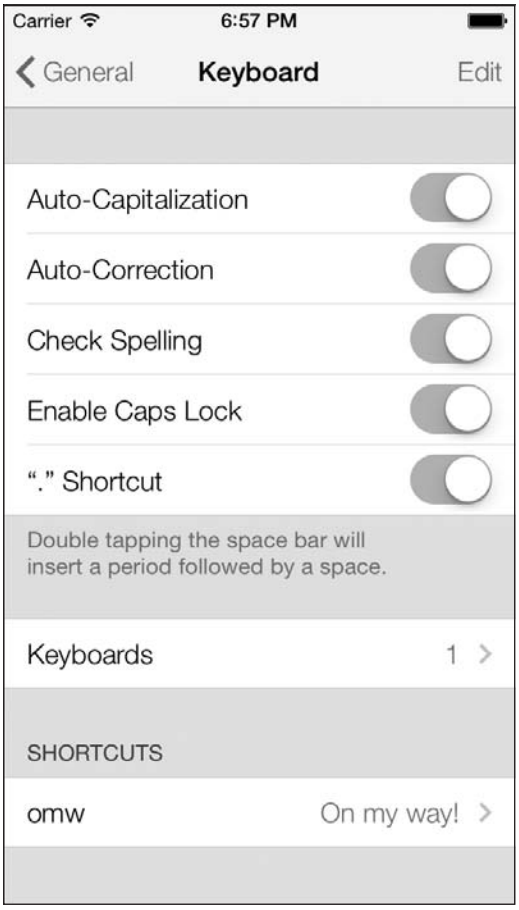


**Рис. 1.7.** Переключатель UISwitch, применяемый в приложении Settings (Настройки) в iPhone

Создать переключатель можно либо с помощью конструктора интерфейса, либо сделав экземпляр такого переключателя в коде. Решим эту задачу вторым способом. Итак, следующая проблема — определить, в каком классе разместить соответствующий код. Это должен быть класс `View Controller` (Контроллер вида), который мы еще не изучали, но, поскольку в этой главе мы создаем программу типа Single View Application (Приложение с единственным видом), файл реализации (`.m`) контроллера вида будет называться `ViewController.m`. Откроем этот файл.

Создадим свойство типа UISwitch и назовем его mainSwitch:

```
#import "ViewController.h"

@interface ViewController ()
@property (nonatomic, strong) UISwitch *mainSwitch;
@end

@implementation ViewController
...
```

Теперь перейдем к файлу реализации контроллера вида (файлу .m) и синтезируем свойство mySwitch:

```
#import "Creating_and_Using_Switches_with_UISwitchViewController.h"

@implementation Creating_and_Using_Switches_with_UISwitchViewController

@synthesize mySwitch;

...
```

Можно продолжить и перейти к созданию переключателя. Найдем метод viewDidLoad в файле реализации нашего контроллера вида:

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

Создадим переключатель и поместим его в виде, в котором находится контроллер нашего вида:

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

  /* Создаем переключатель */
  self.mainSwitch = [[UISwitch alloc] initWithFrame:
                     CGRectMake(100, 100, 0, 0)];

  [self.view addSubview:self.mainSwitch];
}
```

Итак, мы выделили объект типа UISwitch и применили метод initWithFrame: для инициализации переключателя. Обратите внимание: параметр, который мы должны передать этому методу, относится к типу CGRect. CGRect определяет границы прямоугольника, отсчитывая их от точки с координатами ($x$; $y$), находящейся в левом верхнем углу прямоугольника, и также используя данные о его ширине и высоте. Можно создать CGRect, воспользовавшись встраиваемым методом CGRectMake, где первые два параметра, передаваемые методу, — это координаты ($x$; $y$), а следующие два — высота и ширина прямоугольника.

Создав переключатель, мы просто добавляем его к виду нашего контроллера.

Теперь запустим приложение на эмуляторе iPhone. На рис. 1.8 показано, что происходит.

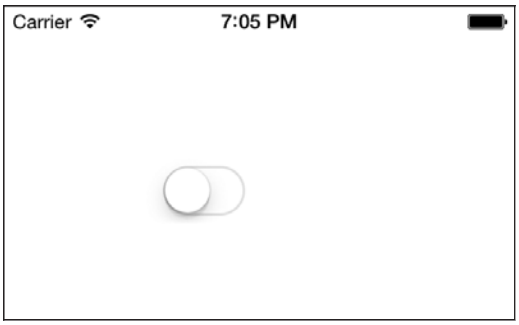


**Рис. 1.8.** Переключатель, размещенный в виде

Как видите, по умолчанию переключатель находится в состоянии Off (Выкл.). Чтобы задать в качестве стандартного противоположное состояние, можно изменить значение свойства `on` экземпляра `UISwitch`. Или можно вызвать метод `setOn:`, относящийся к переключателю:

```
[self.mySwitch setOn:YES];
```

Мы можем немного облегчить работу пользователю, применив метод переключателя `setOn:animated:`. Параметр `animated` принимает логическое значение. Если логическое значение равно `YES`, то при переходе переключателя из состояния `on` в состояние `off` этот процесс будет анимироваться, а также будут анимироваться любые взаимодействия пользователя с переключателем.

Очевидно, вы можете считывать информацию свойства `on` переключателя, чтобы узнавать, включен переключатель в данный момент или выключен. В качестве альтернативы можно пользоваться методом переключателя `isOn`:

```
if ([self.mySwitch isOn]){
  NSLog(@"The switch is on.");
} else {
  NSLog(@"The switch is off.");
}
```

Если вы хотите получать уведомления о том, *когда* переключатель переходит в состояние «включено» или «выключено», необходимо указать ваш класс как *цель* (Target) переключателя, воспользовавшись методом `addTarget:action:forControlEvents:` класса `UISwitch`:

```
[self.mySwitch addTarget:self
                  action:@selector(switchIsChanged:)
        forControlEvents:UIControlEventValueChanged];
```

Затем реализуем метод `switchIsChanged:`. Когда среда времени исполнения вызовет этот метод в ответ на событие переключателя `UIControlEventValueChanged`, она передаст переключатель как параметр данного метода и вы сможете узнать, какой именно переключатель инициировал данное событие:

```
- (void) switchIsChanged:(UISwitch *)paramSender{

  NSLog(@"Sender is = %@", paramSender);
```

```
  if ([paramSender isOn]){
    NSLog(@"The switch is turned on.");
  } else {
    NSLog(@"The switch is turned off.");
  }

}
```

Теперь попробуем запустить наше приложение в эмуляторе iOS. В окне консоли вы увидите примерно такие сообщения:

```
Sender is = <UISwitch: 0x6e13500;
             frame = (100100; 7927);
             layer = <CALayer: 0x6e13700>>
The switch is turned off.
Sender is = <UISwitch: 0x6e13500;
             frame = (100100; 7927);
             layer = <CALayer: 0x6e13700>>
```

Переключатель включен.

# 1.3. Оформление UISwitch

## Постановка задачи

Вы вставили в ваш пользовательский интерфейс несколько экземпляров `UISwitch` и теперь хотите оформить их так, чтобы они вписывались в этот графический интерфейс.

## Решение

Используйте одно из свойств настройки тонов/изображений класса `UISwitch`, например `tintColor` или `onTintColor`.

## Обсуждение

Apple проделала огромную работу по обеспечению оформления компонентов пользовательского интерфейса, в частности `UISwitch`. В предыдущих версиях SDK разработчикам приходилось производить подкласс от `UISwitch` лишь для того, чтобы изменить внешний вид и цвет элемента. В современном iOS SDK такие задачи решаются гораздо проще.

Существует два основных способа оформления переключателя.

- *Работа с оттенками.* Оттенки — это цветовые тона, которые вы можете применять к компоненту пользовательского интерфейса, например к `UISwitch`. Новый оттенок накладывается поверх актуального цвета компонента. Например, при работе с обычным `UISwitch` вы наверняка сталкивались с разными цветами.

Когда вы применяете оттенок поверх цвета, этот цвет смешивается с наложенным оттенком. Таким образом создается разновидность оттенка, действующая в данном элементе пользовательского интерфейса.

- *Изображения*. Переключателю соответствуют:
  - *изображение включенного состояния*. Находится на переключателе, когда он включен. Ширина изображения составляет 77 точек, высота — 22 точки;
  - *изображение выключенного состояния*. Находится на переключателе, когда он выключен. Ширина изображения составляет 77 точек, высота — 22 точки.

На рис. 1.9 показаны примеры изображений, используемых при включенном и выключенном переключателе.

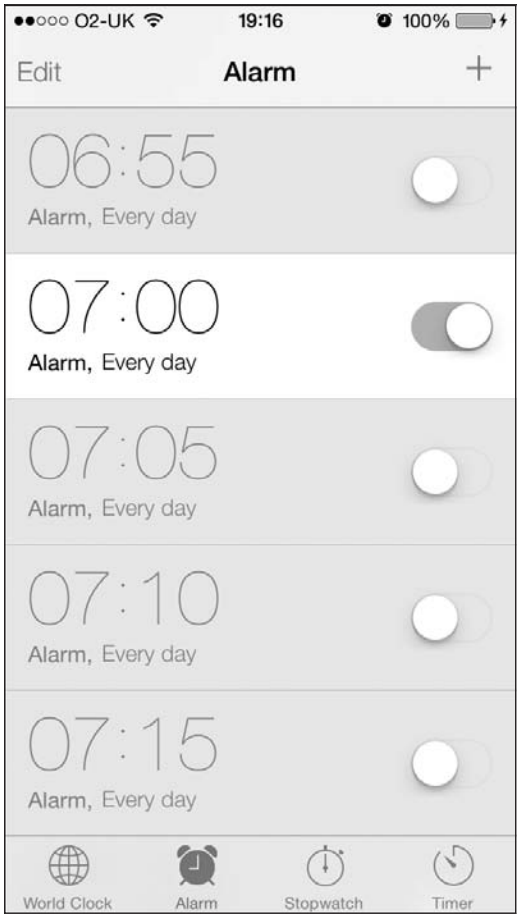


**Рис. 1.9.** Переключатель UISwitch и изображения, соответствующие его включенному и выключенному состояниям

Итак, переключатель может находиться в одном из двух состояний — он либо включен, либо выключен. Теперь рассмотрим, как изменить оттенок переключате-

ля, находящегося в пользовательском интерфейсе. Это можно сделать с помощью трех важных свойств класса UISwitch (все эти свойства относятся к типу UIColor):

- tintColor — оттенок, применяемый к переключателю в выключенном состоянии. К сожалению, Apple подобрала для него не совсем точное название (правильнее было бы, конечно, назвать это свойство offTintColor);
- thumbTintColor — оттенок, который будет применяться к рычажку переключателя;
- onTintColor — оттенок, применяемый к переключателю во включенном состоянии.

Далее приведен простой пример кода, изменяющий оттенок переключателя во включенном состоянии на красный, в выключенном — на коричневый. При этом рычажок будет иметь зеленый цвет. Это не самая лучшая комбинация цветов, но в целях, поставленных в данном разделе, я остановлюсь именно на таком варианте:

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

  /* Создаем переключатель */
  self.mainSwitch = [[UISwitch alloc] initWithFrame:CGRectZero];
  self.mainSwitch.center = self.view.center;
  [self.view addSubview:self.mainSwitch];

  /* Оформляем переключатель */

  /* Изменяем оттенок, который будет у переключателя в выключенном виде */
  self.mainSwitch.tintColor = [UIColor redColor];
  /* Изменяем оттенок, который будет у переключателя во включенном виде */
  self.mainSwitch.onTintColor = [UIColor brownColor];
  /* Изменяем также оттенок рычажка на переключателе */
  self.mainSwitch.thumbTintColor = [UIColor greenColor];
}
```

Теперь, когда мы закончили работу с оттенками переключателя, перейдем к оформлению внешнего вида переключателя, связанному с использованием изображений «включено» и «выключено». При этом не забываем, что заказные изображения «включено» и «выключено» поддерживаются только в iOS 6 и старше. iOS 7 игнорирует такие изображения и при оформлении внешнего вида работает только с оттенками. Как было указано ранее, оба варианта изображения на переключателе — как для включенного, так и для выключенного состояния — должны иметь ширину 77 точек и высоту 22 точки. Поэтому я подготовил новый комплект таких изображений (для работы с обычным и сетчатым дисплеем). Я добавил их в мой проект в Xcode под названиями On@2x.png и Off@2x.png (для сетчатого дисплея), а также поместил здесь разновидности изображений для обычного дисплея. Теперь нам предстоит создать переключатель, но присвоить ему заказные изображения «включено» и «выключено». Для этого воспользуемся следующими свойствами UISwitch:

- `onImage` — как указано ранее, это изображение будет использоваться, когда переключатель включен;
- `offImage` — это изображение соответствует переключателю в состоянии «выключено».

А вот код, позволяющий добиться такого эффекта:

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

  /* Создаем переключатель */
  self.mainSwitch = [[UISwitch alloc] initWithFrame:CGRectZero];
  self.mainSwitch.center = self.view.center;
  /* Убеждаемся, что переключатель не выглядит размытым в iOS-эмуляторе */
  self.mainSwitch.frame = [self roundedValuesInRect:self.mainSwitch.frame];
  [self.view addSubview:self.mainSwitch];

  /* Оформляем переключатель */
  self.mainSwitch.onImage = [UIImage imageNamed:@"On"];
  self.mainSwitch.offImage = [UIImage imageNamed:@"Off"];
}
```

## См. также

Раздел 1.2.

# 1.4. Выбор значений с помощью UIPickerView

## Постановка задачи

Необходимо предоставить пользователю приложения возможность выбирать значения из списка.

## Решение

Воспользуйтесь классом `UIPickerView`.

## Обсуждение

Вид выбора (Picker View) — это элемент графического интерфейса, позволяющий отображать для пользователей списки значений, из которых пользователь затем может выбрать одно. В разделе Timer (Таймер) приложения Clock (Часы) в iPhone мы видим именно такой пример (рис. 1.10).

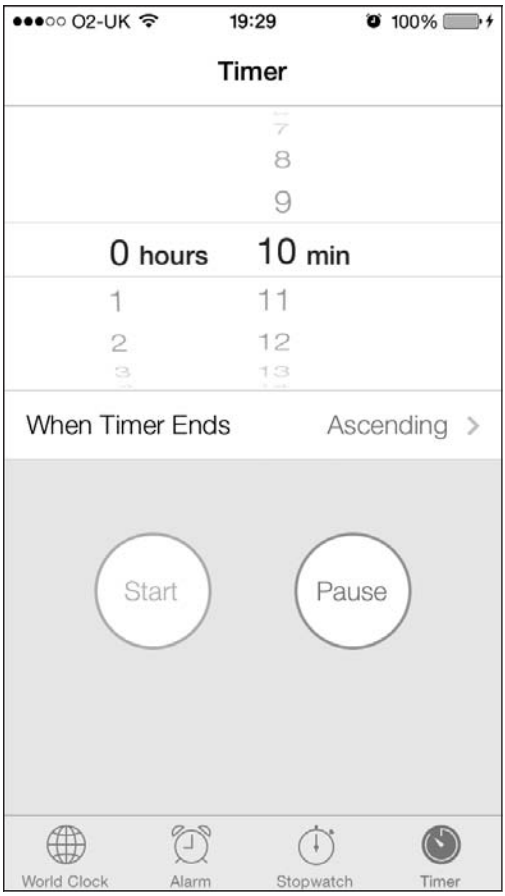


**Рис. 1.10.** Вид выбора, расположенный в верхней части экрана

Как видите, в отдельно взятом виде выбора содержится два независимых визуальных элемента, один слева, другой справа. В левой части вида отображаются часы (0, 1, 2 и т. д.), а в правой — минуты (18, 19, 20, 21, 22 и т. д.). Два этих элемента называются *компонентами*. В каждом компоненте есть строки (Rows). На самом деле любой элемент в любом компоненте представлен строкой, как мы вскоре увидим. Например, в левом компоненте 0 hours — это строка, 1 — это строка и т. д.

Создадим вид выбора в виде нашего контроллера. Если вы не знаете, где находится исходный код того вида, в котором расположен контроллер, обратитесь к разделу 1.2, где обсуждается этот вопрос.

Сначала перейдем к файлу реализации `.m` контроллера нашего вида и определим в нем вид выбора:

```
@interface ViewController ()
@property (nonatomic, strong) UIPickerView *myPicker;
@end
```

```
@implementation ViewController

...
```

А теперь создадим вид выбора в методе `viewDidLoad` контроллера нашего вида:

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

  self.view.backgroundColor = [UIColor whiteColor];

  self.myPicker = [[UIPickerView alloc] init];
  self.myPicker.center = self.view.center;
  [self.view addSubview:self.myPicker];

}
```

В данном примере необходимо отметить, что вид выбора выравнивается по центру того вида, в котором находится. Если мы запустим это приложение в эмуляторе iOS 7, то увидим пустой экран. Дело в том, что в iOS 7 сам элемент для выбора белый и мы видим фон контроллера вида.

Вид выбора отображается в виде сплошного белого поля потому, что мы еще не наполнили его какими-либо значениями. Сделаем это. Итак, нам потребуется указать источник данных для вида выбора, а потом убедиться в том, что контроллер вида соответствует протоколу, требуемому источником данных. Источник данных экземпляра `UIPickerView` должен подчиняться протоколу `UIPickerViewDataSource`, так что обеспечим соответствие данного вида условиям этого протокола в файле `.m`:

```
@interface ViewController () <UIPickerViewDataSource, UIPickerViewDelegate>
@property (nonatomic, strong) UIPickerView *myPicker;
@end

@implementation ViewController

...
```

Хорошо. Теперь изменим наш код в файле реализации, чтобы гарантировать, что актуальный контроллер вида выбран в качестве источника данных для вида выбора:

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

  self.myPicker = [[UIPickerView alloc] init];
  self.myPicker.dataSource = self;
  self.myPicker.center = self.view.center;
  [self.view addSubview:self.myPicker];

}
```

После этого, попытавшись скомпилировать приложение, вы увидите, что компилятор начинает выдавать предупреждения. Эти предупреждения сообщают, что

вы еще не реализовали некоторые методы, внедрения которых требует протокол UIPickerViewDataSource. Чтобы исправить эту ситуацию, нужно нажать Command+Shift+O, ввести UIPickerViewDataSource и нажать Enter. Так вы попадете к тому месту в вашем коде, где определяется данный протокол, и увидите нечто подобное:

```
@protocol UIPickerViewDataSource<NSObject>
@required

// Возвращает количество столбцов для отображения
- (NSInteger)numberOfComponentsInPickerView:(UIPickerView *)pickerView;

// Возвращает количество строк в каждом компоненте
- (NSInteger)pickerView:(UIPickerView *)pickerView
numberOfRowsInComponent:(NSInteger)component;
@end
```

Вы заметили здесь ключевое слово @required? Оно означает, что любой класс, желающий стать источником данных для вида выбора, *обязан* реализовывать эти методы. Напишем их в файле реализации контроллера нашего вида:

```
- (NSInteger)numberOfComponentsInPickerView:(UIPickerView *)pickerView{

  if ([pickerView isEqual:self.myPicker]){
    return 1;
  }

  return 0;

}
- (NSInteger) pickerView:(UIPickerView *)pickerView
numberOfRowsInComponent:(NSInteger)component{

  if ([pickerView isEqual:self.myPicker]){
    return 10;
  }

  return 0;
}
```

Итак, что здесь происходит? Рассмотрим, какие данные предполагает каждый из методов источника:

- numberOfComponentsInPickerView: — этот метод передает объект вида выбора в качестве параметра, а в качестве возвращаемого значения ожидает целое число, указывающее, сколько компонентов вы хотели бы отобразить в этом виде выбора;
- pickerView:numberOfRowsInComponent: — для каждого компонента, добавляемого в вид выбора, необходимо указать системе, какое количество строк вы хотите отобразить в данном компоненте. Этот метод передает вам экземпляр вида выбора, а в качестве возвращаемого значения ожидает целое число, сообщающее среде времени исполнения, сколько строк вы хотели бы отобразить в этом компоненте.

Итак, мы приказываем системе отобразить один компонент всего с 10 строками для вида выбора, который мы создали ранее и назвали `myPicker`.

Скомпилируйте приложение и запустите его в эмуляторе iPhone (рис. 1.11). Хм-м-м, и что же это?

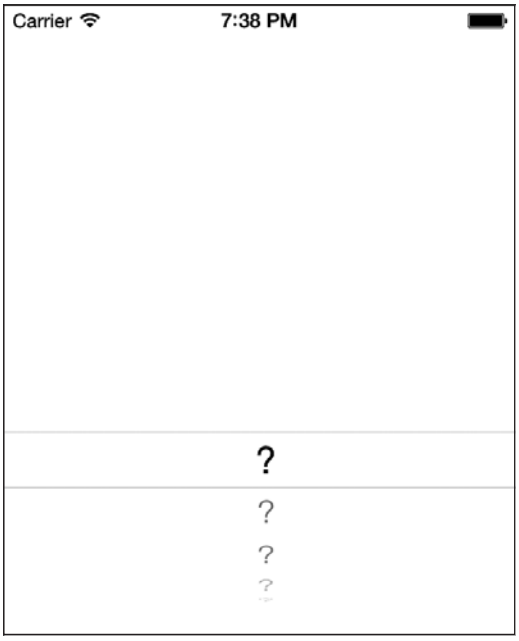


**Рис. 1.11.** Вот как выглядит вид выбора, когда неизвестно, какую информацию в нем отображать

По всей видимости, наш вид выбора знает, сколько компонентов в нем должно быть и сколько строк он должен отображать в интересующем нас компоненте, но не знает, какой *текст* должен содержаться в каждой строке. Этот вопрос обязательно следует прояснить, и мы решим данную проблему, предоставив делегат для вида выбора. Делегат экземпляра `UIPickerView` должен подчиняться протоколу `UIPickerViewDelegate` и реализовывать все методы, помеченные как `@required`.

Нас интересует только один метод делегата `UIPickerViewDelegate`, а именно `pickerView:titleForRow:forComponent:`. Этот метод передает вам индекс актуального раздела и индекс актуальной строки в данном разделе вида выбора и в качестве возвращаемого значения ожидает экземпляр `NSString`. Строка, представленная `NSString`, отобразится в заданном ряду внутри компонента. В рассматриваемом случае я предпочитаю просто отобразить первую строку как «Строка 1», а затем продолжить: «Строка 2», «Строка 3» и т. д. Не забывайте, что потребуется также установить свойство `delegate` нашего вида выбора:

```
self.myPicker.delegate = self;
```

А теперь обработаем только что изученный метод делегата:

```
- (NSString *)pickerView:(UIPickerView *)pickerView
             titleForRow:(NSInteger)row
```

```
            forComponent:(NSInteger)component{

  if ([pickerView isEqual:self.myPicker]){

    /* Строка имеет нулевое основание, а мы хотим, чтобы первая строка
    (с индексом 0) отображалась как строка 1. Таким образом, нужно
    прибавить +1 к индексу каждой строки. */
    result = [NSString stringWithFormat:@"Row %ld", (long)row + 1];

  }
  return nil;
}
```

Теперь запустим приложение и посмотрим, что происходит (рис. 1.12).

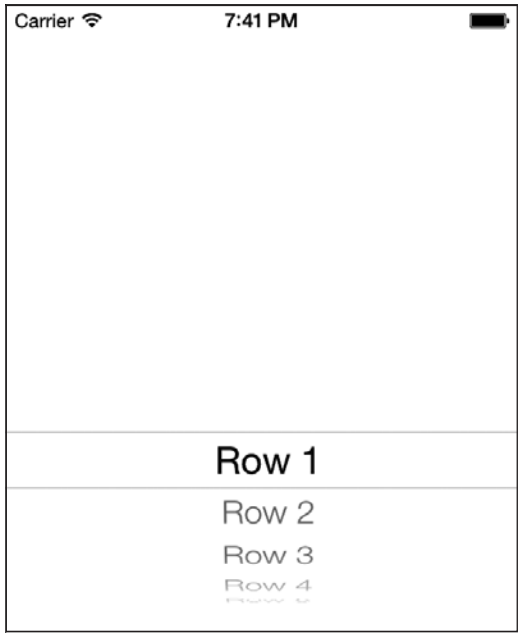


**Рис. 1.12.** Вид выбора с одним разделом и несколькими строками

Виды с возможностью выбора в iOS 6 и старше могут подсвечивать выбранный вариант с помощью свойства `showsSelectionIndicator`, по умолчанию имеющего значение `NO`. Вы можете либо напрямую изменить значение этого свойства на `YES`, либо воспользоваться методом `setShowsSelectionIndicator:` вида выбора, чтобы включить этот индикатор:

```
self.myPicker.showsSelectionIndicator = YES;
```

Снова предположим, что мы создаем вид выбора в окончательной версии нашего приложения. Какая польза от вида выбора, если мы не можем определить, что именно пользователь выбрал в каждом из компонентов? Да, хорошо, что Apple уже позаботилась о решении этой проблемы и предоставила нам возможность спрашивать

вид выбора о выбранном варианте. Вызовем метод `selectedRowInComponent:` класса `UIPickerView` и передадим индекс компонента (с нулевым основанием), а в качестве возвращаемого значения получим целое число. Это число будет представлять собой индекс с нулевым основанием, сообщающий строку, которая в данный момент выбрана в интересующем нас компоненте.

Если во время исполнения вам потребуется изменить значения, содержащиеся в вашем виде выбора, необходимо гарантировать, что вид выбора сможет перегружать данные, заменяя старую информацию новой, получаемой из источника и от делегата. Для этого нужно либо принудительно заставить все компоненты перезагрузить содержащиеся в них данные (это делается с помощью метода `reloadAllComponents`), либо приказать конкретному компоненту перезагрузить содержащиеся в нем данные. Во втором случае применяется метод `reloadComponent:`. Ему передается индекс компонента, который необходимо перезагрузить.

## См. также

Раздел 1.2.

# 1.5. Выбор даты и времени с помощью UIDatePicker

## Постановка задачи

Необходимо предоставить пользователям вашего приложения возможность выбирать дату и время. Для этого нужен интуитивно понятный и уже готовый пользовательский интерфейс.

## Решение

Воспользуйтесь классом `UIDatePicker`.

## Обсуждение

Класс `UIDatePicker` очень напоминает класс `UIPickerView`. Фактически `UIDatePicker` — это уже заполненный вид выбора. Хорошим примером такого вида является программа Calendar (Календарь) в iPhone (рис. 1.13).

Для начала объявим свойство типа `UIDatePicker`, а потом выделим и инициализируем это свойство и добавим его в вид, в котором находится контроллер нашего вида:

```
#import "ViewController.h"

@interface ViewController ()
@property (nonatomic, strong) UIDatePicker *myDatePicker;
```

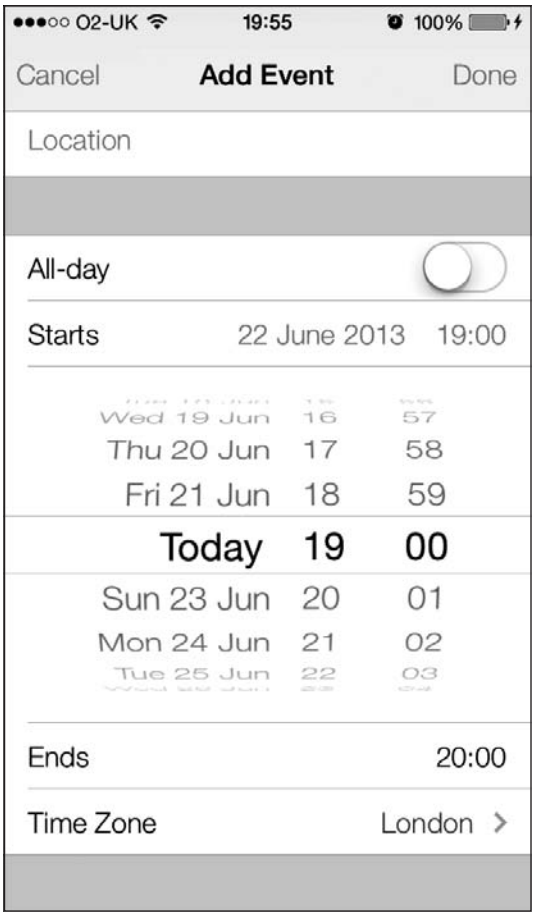


**Рис. 1.13.** Вид для выбора даты показан в нижней части экрана

```
@end

@implementation ViewController

...
```

А теперь, как и планировалось, инстанцируем вид для выбора даты:

```
- (void)viewDidLoad{
  [super viewDidLoad];
  self.myDatePicker = [[UIDatePicker alloc] init];
  self.myDatePicker.center = self.view.center;
  [self.view addSubview:self.myDatePicker];
}
```

После этого запустим приложение и посмотрим, как оно выглядит (рис. 1.14).

Как видите, по умолчанию в виде выбора даты ставится сегодняшняя дата. Начиная работать с такими инструментами, первым делом нужно уяснить, что они


**Рис. 1.14.** Простой вид для выбора даты

могут иметь различные стили оформления и режимы работы. Режим можно изменить, работая со свойством `datePickerMode`, тип которого — `UIDatePickerMode`:

```
typedef enum {
  UIDatePickerModeTime,
  UIDatePickerModeDate,
  UIDatePickerModeDateAndTime,
  UIDatePickerModeCountDownTimer,
} UIDatePickerMode;
```

В зависимости от конкретной задачи, стоящей перед вами, для режима вида выбора даты можно задать любое из значений, перечисленных в списке `UIDatePickerMode`. Далее по мере обсуждения данной темы мы рассмотрим некоторые из этих значений.

Теперь, когда вы успешно смогли отобразить на экране вид для выбора даты, можно попытаться получить дату, которая выведена в нем в настоящий момент. Для получения этой информации используется свойство `date` данного вида. Другой способ — применить метод `date` к виду выбора даты:

```
NSDate *currentDate = self.myDatePicker.date;
NSLog(@"Date = %@", currentDate);
```

Подобно классу `UISwitch`, вид для выбора даты также посылает своим целям инициирующие сообщения (Action Messages) всякий раз, когда отображаемая в виде дата изменяется. Чтобы иметь возможность реагировать на эти сообщения, получатель должен добавить себя в список целей вида выбора даты. Для этого используется метод `addTarget:action:forControlEvents:` следующим образом:

```
- (void) datePickerDateChanged:(UIDatePicker *)paramDatePicker{

  if ([paramDatePicker isEqual:self.myDatePicker]){
    NSLog(@"Selected date = %@", paramDatePicker.date);
  }

}

- (void)viewDidLoad{
  [super viewDidLoad];
  self.myDatePicker = [[UIDatePicker alloc] init];
  self.myDatePicker.center = self.view.center;
  [self.view addSubview:self.myDatePicker];
  [self.myDatePicker addTarget:self
                        action:@selector(datePickerDateChanged:)
              forControlEvents:UIControlEventValueChanged];

}
```

Теперь всякий раз, когда пользователь изменяет дату, вы будете получать сообщение от вида выбора даты.

Пользуясь видом для выбора даты, можно задавать минимальную и максимальную даты, которые он способен отображать. Для этого сначала нужно переключить вид выбора даты в режим `UIDatePickerModeDate`, а потом с помощью свойств `maximumDate` и `minimumDate` откорректировать этот диапазон:

```
- (void)viewDidLoad{
  [super viewDidLoad];
  self.myDatePicker = [[UIDatePicker alloc] init];
  self.myDatePicker.center = self.view.center;
  self.myDatePicker.datePickerMode = UIDatePickerModeDate;
  [self.view addSubview:self.myDatePicker];

  NSTimeInterval oneYearTime = 365 * 24 * 60 * 60;
  NSDate *todayDate = [NSDate date];

  NSDate *oneYearFromToday = [todayDate
                              dateByAddingTimeInterval:oneYearTime];

  NSDate *twoYearsFromToday = [todayDate
                               dateByAddingTimeInterval:2 * oneYearTime];

  self.myDatePicker.minimumDate = oneYearFromToday;
  self.myDatePicker.maximumDate = twoYearsFromToday;
}
```

Применяя два этих свойства, можно ограничить доступный пользователю диапазон выбора даты конкретными пределами, как показано на рис. 1.15. В приведенном образце кода мы позволяем пользователю задавать даты в диапазоне от года до двух лет, отсчитывая с настоящего момента.


**Рис. 1.15.** Минимальная и максимальная даты при работе с видом выбора даты

Если вы хотите применять вид выбора даты в качестве таймера обратного отсчета, нужно задать для этого вида режим `UIDatePickerModeCountDownTimer` и использовать свойство `countDownDuration` вида выбора даты для указания длительности обратного отсчета, задаваемой по умолчанию. Например, если вы желаете предложить пользователю такой таймер и задать в качестве периода ведения обратного отсчета 2 минуты, нужно написать следующий код:

```
- (void)viewDidLoad{
  [super viewDidLoad];
  self.view.backgroundColor = [UIColor whiteColor];
  self.myDatePicker = [[UIDatePicker alloc] init];
  self.myDatePicker.center = self.view.center;
  self.myDatePicker.datePickerMode = UIDatePickerModeCountDownTimer;
  [self.view addSubview:self.myDatePicker];
  NSTimeInterval twoMinutes = 2 * 60;
  [self.myDatePicker setCountDownDuration:twoMinutes];
}
```

Результат показан на рис. 1.16.

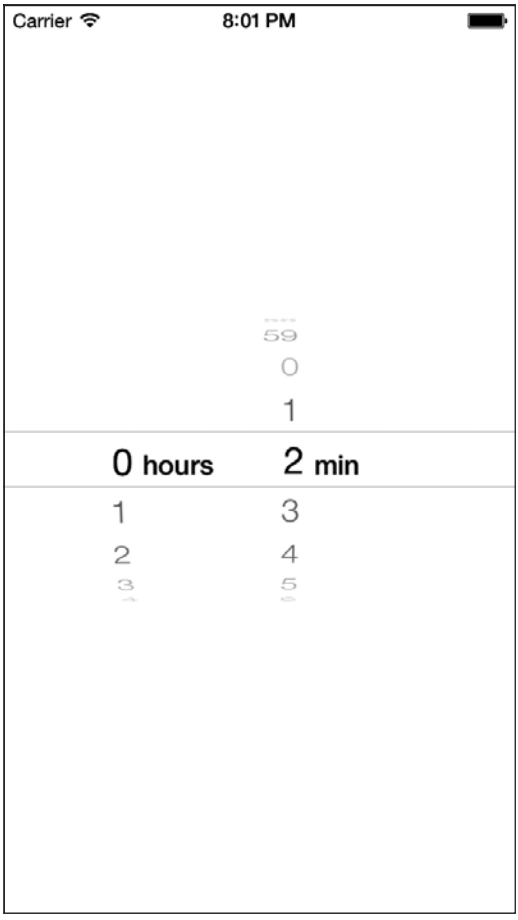


**Рис. 1.16.** Таймер обратного отсчета в виде для выбора даты, где стандартная длительность обратного отсчета равна 2 минутам

# 1.6. Реализация инструмента для выбора временных рамок с помощью UISlider

## Постановка задачи

Необходимо дать пользователям возможность указывать определенное значение из диапазона и предоставить для этого удобный в применении и интуитивно понятный пользовательский интерфейс.

## Решение

Используйте класс `UISlider`.

## Обсуждение

Вероятно, вы уже знаете, что такое слайдер. Пример слайдера показан на рис. 1.17.


**Рис. 1.17.** В нижней части экрана находится слайдер, регулирующий уровень громкости

Чтобы создать слайдер, нужно инстанцировать объект типа `UISlider`. Создадим слайдер и поместим его на вид нашего контроллера. Начнем с файла реализации нашего контроллера вида:

```
#import "ViewController.h"

@interface ViewController ()
```

```
@property (nonatomic, strong) UISlider *slider;
@end

@implementation ViewController

...
```

Теперь рассмотрим метод `viewDidLoad` и создадим сам компонент-слайдер. В этом коде мы будем создавать слайдер, позволяющий выбирать значения в диапазоне от 0 до 100. По умолчанию ползунок слайдера будет установлен в среднем значении шкалы:

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

  self.slider = [[UISlider alloc] initWithFrame:CGRectMake(0.0f,
                                                           0.0f,
                                                           200.0f,
                                                           23.0f)];
  self.slider.center = self.view.center;
  self.slider.minimumValue = 0.0f;
  self.slider.maximumValue = 100.0f;
  self.slider.value = self.slider.maximumValue / 2.0;
  [self.view addSubview:self.slider];
}
```

Диапазон слайдера совершенно не зависит от его внешнего вида. С помощью указателя диапазона мы приказываем слайдеру вычислить его (слайдера) значение, основываясь на относительной позиции в рамках диапазона. Например, если для слайдера задан диапазон от 0 до 100, то когда ползунок слайдера расположен у левого края шкалы, свойство слайдера value равно 0, а если ползунок стоит у правого края, оно равно 100.

Как будут выглядеть результаты? Теперь вы можете запустить приложение в эмуляторе (рис. 1.18).

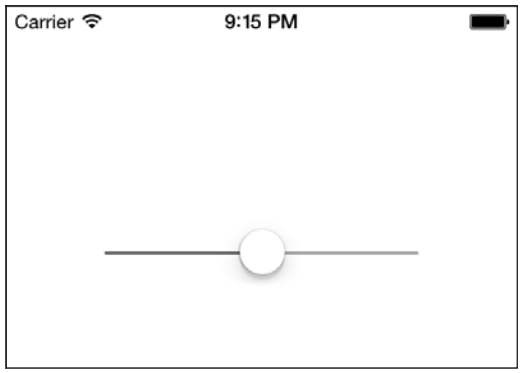


**Рис. 1.18.** Обычный слайдер в центре экрана

Для получения желаемых результатов мы использовали несколько свойств слайдера. Что это за свойства?

- `minimumValue` — задает минимальное значение диапазона, поддерживаемого слайдером.
- `maximumValue` — задает максимальное значение диапазона, поддерживаемого слайдером.
- `value` — текущее значение слайдера. Это свойство доступно как для чтения, так и для изменения, то есть вы можете как считывать это значение, так и записывать в него информацию. Если вы хотите, чтобы при перемещении ползунка в это значение включалась анимация, то можно вызвать метод слайдера `setValue:animated:` и передать `YES` в качестве значения параметра `animated`.

Ползунок слайдера называется также бегунком. Если вы хотите получать событие всякий раз, когда передвигается ползунок слайдера, нужно добавить ваш объект, которому требуется информация о таких событиях, в качестве цели слайдера с помощью относящегося к слайдеру метода `addTarget:action:forControlEvents:`:

```
- (void) sliderValueChanged:(UISlider *)paramSender{

  if ([paramSender isEqual:self.mySlider]){
    NSLog(@"New value = %f", paramSender.value);
  }

}
- (void)viewDidLoad{
  [super viewDidLoad];
  self.view.backgroundColor = [UIColor whiteColor];
  self.mySlider = [[UISlider alloc] initWithFrame:CGRectMake(0.0f,
                                                             0.0f,
                                                             200.0f,
                                                             23.0f)];
  self.slider.center = self.view.center;
  self.slider.minimumValue = 0.0f;
  self.slider.maximumValue = 100.0f;
  self.slider.value = self.slider.maximumValue / 2.0;
  [self.view addSubview:self.slider];

  [self.slider addTarget:self
                  action:@selector(sliderValueChanged:)
        forControlEvents:UIControlEventValueChanged];
}
```

Если сейчас запустить приложение в эмуляторе, вы увидите, что вызывается целевой метод `sliderValueChanged:` и это происходит *всякий раз, как только* перемещается ползунок слайдера. Возможно, именно этого вы и хотели. Но в некоторых случаях уведомление требуется лишь тогда, когда пользователь отпустил ползунок, установив его в новом значении. Если вы хотите дождаться такого уведомления, установите для свойства слайдера `continuous` значение `NO`. Если это свойство имеет значение `YES` (задаваемое по умолчанию), то на цели слайдера вызов будет идти непрерывно все то время, пока движется ползунок.

В SDK iOS разработчик также может изменять внешний вид слайдера. Например, ползунок может иметь нестандартный вид. Чтобы изменить внешний вид ползунка, просто пользуйтесь методом `setThumbImage:forState:` и передавайте нужное изображение, а также второй параметр, который может принимать одно из двух значений:

- `UIControlStateNormal` — обычное состояние ползунка, когда его не трогает пользователь;
- `UIControlStateHighlighted` — изображение, которое должно быть на месте ползунка, когда пользователь начинает его двигать.

Я подготовил два изображения: одно для стандартного состояния ползунка, а другое — для активного (затронутого) состояния. Добавим их к слайдеру:

```
[self.slider setThumbImage:[UIImage imageNamed:@"ThumbNormal.png"]
                  forState:UIControlStateNormal];
[self.slider setThumbImage:[UIImage imageNamed:@"ThumbHighlighted.png"]
                  forState:UIControlStateHighlighted];
```

Теперь взглянем, как выглядит в эмуляторе неактивный слайдер (рис. 1.19).

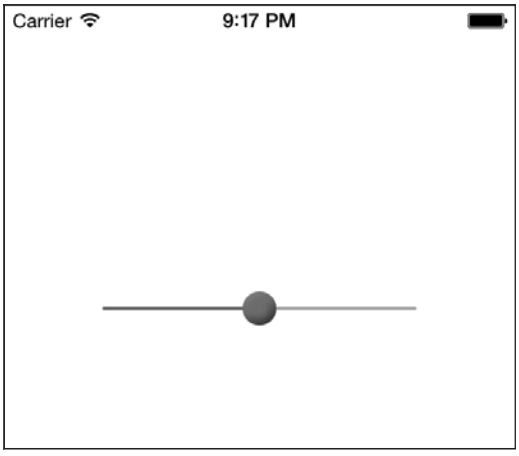


**Рис. 1.19.** Слайдер со специально оформленным ползунком

# 1.7. Оформление UISlider

## Постановка задачи

Вы используете компонент графического интерфейса `UISlider`, оформленный по умолчанию, и хотите на свой вкус изменить его внешний вид.

## Решение

Либо измените оттенки различных частей слайдера, либо подготовьте для его элементов собственные рисунки.

## Обсуждение

Apple проделала огромную работу, предоставив нам в iOS SDK методы для оформления компонентов пользовательского интерфейса. В частности, оформление может быть связано с изменением оттенков различных частей компонента в интерфейсе. Схема, демонстрирующая компонентный состав пользовательского интерфейса, приведена на рис. 1.20.

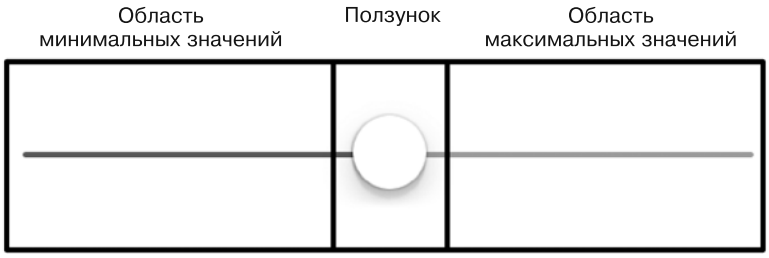


**Рис. 1.20.** Различные компоненты UISlider

Для каждого из этих компонентов UISlider существуют метод и свойство, позволяющие изменять внешний вид слайдера. Простейшими из этих свойств являются те, которые позволяют изменять оттенок соответствующего компонента:

- `minimumTrackTintColor` — это свойство задает оттенок для области минимальных значений;
- `thumbTintColor` — это свойство, как понятно из его названия, задает цвет ползунка;
- `maximumTrackTintColor` — это свойство задает оттенок для области максимальных значений.

Все эти свойства относятся к типу `UIColor`.

В следующем образце кода мы инстанцируем `UISlider` и помещаем его в центре вида, расположенного в нашем контроллере. Кроме того, здесь мы задаем цвет для области минимальных значений (красный), цвет ползунка (черный) и цвет области максимальных значений (зеленый):

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

  /* Создаем слайдер */
  self.slider = [[UISlider alloc] initWithFrame:CGRectMake(0.0f,
                                                           0.0f,
                                                           118.0f,
                                                           23.0f)];
  self.slider.value = 0.5;
  self.slider.minimumValue = 0.0f;
  self.slider.maximumValue = 1.0f;
  self.slider.center = self.view.center;
  [self.view addSubview:self.slider];

  /* Задаем оттенок для области минимальных значений */
  self.slider.minimumTrackTintColor = [UIColor redColor];
```

```
  /* Задаем оттенок для ползунка */
  self.slider.maximumTrackTintColor = [UIColor greenColor];

  /* Задаем цвет для области максимальных значений */
  self.slider.thumbTintColor = [UIColor blackColor];
}
```

Если вы теперь запустите получившееся приложение, то увидите примерно такую картину, как на рис. 1.21.

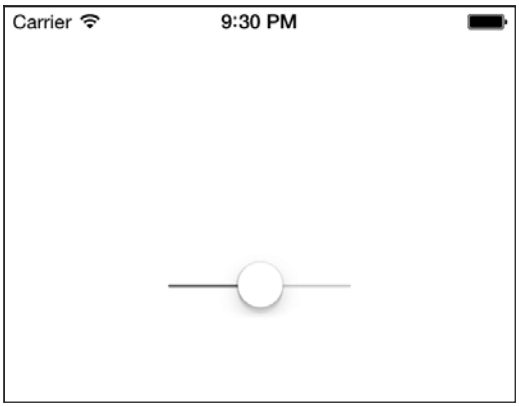


**Рис. 1.21.** Оттенки всех составных частей слайдера изменены

Иногда вам может потребоваться более полный контроль над тем, как слайдер будет выглядеть на экране. При этом одних только оттенков может быть недостаточно. Именно поэтому Apple предлагает и другие способы изменения внешнего вида слайдера, позволяя вам задавать изображения для различных его компонентов. Речь идет о следующих изображениях.

- *Изображение для минимального значения*. Это изображение, которое будет находиться за пределами слайдера у его левого края. По умолчанию такое изображение не предоставляется, поэтому вы его и не увидите, если просто создадите в виде новый слайдер. Вы можете задать такое изображение, чтобы подсказывать пользователю, как трактуется минимальное значение в контексте данного слайдера. Например, в приложении, с помощью которого пользователь может увеличивать или уменьшать яркость экрана, минимальному значению может соответствовать картинка в виде потухшей лампочки. Она показывает, что чем дальше пользователь будет перемещать ползунок в сторону минимального значения (влево), тем более тусклым будет становиться экран. Чтобы изменить это изображение, воспользуйтесь относящимся к слайдеру методом экземпляра `setMinimumValueImage:`. Это изображение должно иметь по 23 точки в высоту и в ширину. При работе с сетчатым дисплеем используйте такое же изображение, только вдвое крупнее.

- *Изображение для области минимальных значений*. Это изображение, которое будет соответствовать колее слайдера левее от ползунка. Чтобы изменить это изображение, воспользуйтесь относящимся к слайдеру методом экземпляра `setMinimumTrackImage:forState:`. Это изображение должно иметь 11 точек в ширину и 9 точек в высоту и допускать изменение размера (подробнее о таких изображениях см. в разделе 17.5).
- *Изображение для ползунка*. Изображение ползунка — это единственный движущийся элемент слайдера. Чтобы изменить это изображение, воспользуйтесь относящимся к слайдеру методом экземпляра `setThumbImage:forState:`. Это изображение должно иметь 23 точки в высоту и 23 точки в ширину.
- *Изображение для области максимальных значений*. Это изображение будет соответствовать той части колеи слайдера, которая находится справа от ползунка. Чтобы изменить это изображение, воспользуйтесь относящимся к слайдеру методом экземпляра `setMaximumTrackImage:forState:`. Это изображение должно иметь 11 точек в ширину и 9 точек в высоту и допускать изменение размера (подробнее о таких изображениях см. в разделе 17.5).
- *Изображение для максимального значения*. Это изображение, которое будет находиться у правого края слайдера. Оно должно напоминать изображение, соответствующее минимальному значению, но, разумеется, трактуется противоположным образом. Вернувшись к примеру с яркостью лампочки, допустим, что справа от колеи с ползунком у нас изображена яркая лампочка, испускающая лучи. Так пользователю будет понятно, что чем дальше вправо он передвигает ползунок, тем ярче становится экран. Чтобы изменить это изображение, воспользуйтесь относящимся к слайдеру методом экземпляра `setMaximumValueImage:`. Это изображение должно иметь 23 точки в высоту и столько же в ширину.

Изображения, предоставляемые вами для областей максимальных и минимальных значений, должны при необходимости изменять размер. Подробнее о таких изображениях рассказано в разделе 17.5.

Для этого упражнения я создал пять уникальных изображений — по одному для каждого компонента слайдера. Убедился, что изображения для областей с максимальными и минимальными значениями поддерживают изменение размера. Оформляя этот слайдер по своему усмотрению, я стремлюсь создать у пользователя впечатление, что он меняет температуру в комнате: при перемещении ползунка влево становится прохладнее, вправо — теплее. Далее приведен код, создающий слайдер и оформляющий различные его компоненты:

```
#import "ViewController.h"

@interface ViewController ()
@property (nonatomic, strong) UISlider *slider;
@end

@implementation ViewController
```

```
/*
Этот метод возвращает изображение переменного размера для области слайдера, содер-
жащей минимальные значения
*/
- (UIImage *) minimumTrackImage{
  UIImage *result = [UIImage imageNamed:@"MinimumTrack"];
  UIEdgeInsets edgeInsets;
  edgeInsets.left = 4.0f;
  edgeInsets.top = 0.0f;
  edgeInsets.right = 0.0f;
  edgeInsets.bottom = 0.0f;
  result = [result resizableImageWithCapInsets:edgeInsets];
  return result;
}

/*
Аналогично предыдущему методу этот возвращает изображение переменного размера
для области слайдера, содержащей максимальные значения
*/
- (UIImage *) maximumTrackImage{
  UIImage *result = [UIImage imageNamed:@"MaximumTrack"];
  UIEdgeInsets edgeInsets;
  edgeInsets.left = 0.0f;
  edgeInsets.top = 0.0f;
  edgeInsets.right = 3.0f;
  edgeInsets.bottom = 0.0f;
  result = [result resizableImageWithCapInsets:edgeInsets];
  return result;
}

- (void)viewDidLoad{
  [super viewDidLoad];

  /* Создаем слайдер */
  self.slider = [[UISlider alloc] initWithFrame:CGRectMake(0.0f,
                                                           0.0f,
                                                           218.0f,
                                                           23.0f)];
  self.slider.value = 0.5;
  self.slider.minimumValue = 0.0f;
  self.slider.maximumValue = 1.0f;
  self.slider.center = self.view.center;
  [self.view addSubview:self.slider];

  /* Изменяем изображение для минимального значения */
  [self.slider setMinimumValueImage:[UIImage imageNamed:@"MinimumValue"]];

  /* Изменяем изображение для области минимальных значений */
  [self.slider setMinimumTrackImage:[self minimumTrackImage]
                           forState:UIControlStateNormal];
```

```
    /* Изменяем изображение ползунка для обоих возможных состояний ползунка: когда
    пользователь его касается и когда не касается */
    [self.slider setThumbImage:[UIImage imageNamed:@"Thumb"]
                      forState:UIControlStateNormal];
    [self.slider setThumbImage:[UIImage imageNamed:@"Thumb"]
                      forState:UIControlStateHighlighted];

    /* Изменяем изображение для области максимальных значений */
    [self.slider setMaximumTrackImage:[self maximumTrackImage]
                             forState:UIControlStateNormal];

    /* Изменяем изображение, соответствующее максимальному значению */
    [self.slider setMaximumValueImage:[UIImage imageNamed:@"MaximumValue"]];
}
```

Ползунок в iOS 7 выглядит совершенно иначе, нежели в более ранних версиях. Как вы догадываетесь, этот элемент стал очень прямолинейным и тонким на вид. Высота максимальной и минимальной отметок на шкале в iOS 7 составляет всего 1 точку, поэтому задавать для этих элементов специальные изображения абсолютно бесполезно — скорее всего, получится некрасиво. Поэтому для оформления этих элементов UISlider в iOS 7 рекомендуется оперировать лишь оттенками, но не присваивать элементу никаких изображений.

## См. также

Раздел 1.6.

# 1.8. Группирование компактных параметров с помощью UISegmentedControl

## Постановка задачи

Требуется предложить пользователям на выбор несколько параметров, из которых они могут выбирать. Пользовательский интерфейс должен оставаться компактным, простым и легким для понимания.

## Решение

Используйте класс `UISegmentedControl`. Пример работы с этим классом показан на рис. 1.22.

## Обсуждение

Сегментированный элемент управления — это сущность, позволяющая отображать в компактном пользовательском интерфейсе наборы параметров, из которых поль-


**Рис. 1.22.** Сегментированный элемент управления, в котором отображаются четыре параметра

зователь может выбирать нужный. Чтобы отобразить сегментированный элемент управления, создайте экземпляр класса `UISegmentedControl`. Начинаем работу с файла реализации (`.m`) нашего контроллера вида:

```
#import "ViewController.h"

@interface ViewController ()
@property (nonatomic, strong) UISegmentedControl *mySegmentedControl;
@end

@implementation ViewController

...
```

Создаем сегментированный элемент управления в методе `viewDidLoad` контроллера нашего вида:

```
- (void)viewDidLoad{
  [super viewDidLoad];
  NSArray *segments = [[NSArray alloc] initWithObjects:
                       @"iPhone",
                       @"iPad",
                       @"iPod",
                       @"iMac", nil];

  self.mySegmentedControl = [[UISegmentedControl alloc]
                             initWithItems:segments];
  self.mySegmentedControl.center = self.view.center;
  [self.view addSubview:self.mySegmentedControl];

}
```

Чтобы представить разные параметры, которые будут предлагаться на выбор в нашем сегментированном элементе управления, мы используем обычный массив строк. Такой элемент управления инициализируется с помощью метода `initWithObjects:`.

Потом передаем сегментированному элементу управления массив строк и изображений. Результат будет как на рис. 1.22.

Теперь пользователь может выбрать в сегментированном элементе управления *один* из параметров. Допустим, он выбирает iPad. Тогда пользовательский интерфейс сегментированного элемента управления изменится и покажет пользователю, какой параметр будет выбран. Получится такое изображение, как на рис. 1.23.

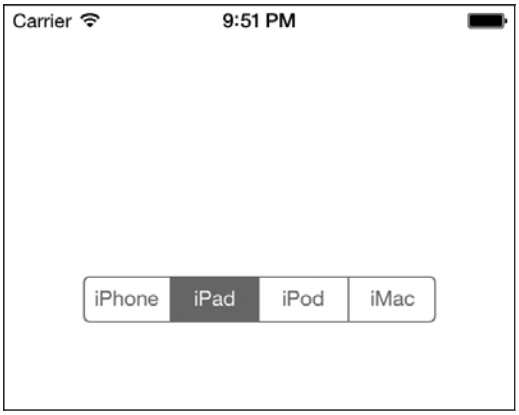


**Рис. 1.23.** Пользователь выбрал один из вариантов в сегментированном элементе управления

Возникает вопрос: как узнать, что пользователь выбрал в сегментированном элементе управления новый параметр? Ответ прост. Как и при работе с `UISwitch` или `UISlider`, применяется метод `addTarget:action:forControlEvents:` сегментированного элемента управления, к которому добавляется цель. Для параметра `forControlEvents` нужно задать значение `UIControlEventValueChanged`, так как именно это событие запускается, когда пользователь выбирает в сегментированном элементе управления новый параметр:

```
- (void) segmentChanged:(UISegmentedControl *)paramSender{
  if ([paramSender isEqual:self.mySegmentedControl]){
    NSInteger selectedSegmentIndex = [paramSender selectedSegmentIndex];

    NSString *selectedSegmentText =
    [paramSender titleForSegmentAtIndex:selectedSegmentIndex];
    NSLog(@"Segment %ld with %@ text is selected",
    (long)selectedSegmentIndex,
    selectedSegmentText);
  }
}

- (void)viewDidLoad{
  [super viewDidLoad];
  NSArray *segments = [[NSArray alloc] initWithObjects:
                       @"iPhone",
                       @"iPad",
```

```
                        @"iPod",
                        @"iMac", nil];

  self.mySegmentedControl = [[UISegmentedControl alloc]
                             initWithItems:segments];
  self.mySegmentedControl.center = self.view.center;
  [self.view addSubview:self.mySegmentedControl];

  [self.mySegmentedControl addTarget:self
                              action:@selector(segmentChanged:)
                    forControlEvents:UIControlEventValueChanged];
}
```

Если пользователь начинает выбирать слева и выбирает каждый параметр (см. рис. 1.22) до правого края, на консоль будет выведен следующий текст:

```
Segment 0 with iPhone text is selected
Segment 1 with iPad text is selected
Segment 2 with iPod text is selected
Segment 3 with iMac text is selected
```

Как видите, мы использовали метод `selectedSegmentIndex` сегментированного элемента управления, чтобы найти индекс варианта, выбранного в настоящий момент. Если ни один из элементов не выбран, метод возвращает значение `-1`. Кроме того, мы использовали метод `titleForSegmentAtIndex:`. Просто передаем этому методу индекс параметра, выбранного в сегментированном элементе управления, а сегментированный элемент управления возвратит текст, соответствующий этому параметру. Ведь просто, правда?

Как вы, вероятно, заметили, как только пользователь отмечает один из параметров в сегментированном элементе управления, этот параметр выбирается и *остается* выбранным, как показано на рис. 1.23. Если вы хотите, чтобы пользователь выбрал параметр, но кнопка этого параметра не оставалась нажатой, а возвращалась к исходной форме (так сказать, «отщелкивалась обратно», как и обычная кнопка), то нужно задать для свойства `momentary` сегментированного элемента управления значение `YES`:

```
self.mySegmentedControl.momentary = YES;
```

Одна из самых приятных особенностей сегментированных элементов управления заключается в том, что они могут содержать не только текст, но и изображения. Для этого нужно просто использовать метод-инициализатор `initWithObjects:` класса `UISegmentedControl` и передать с этим методом те строки и изображения, которые будут применяться при реализации соответствующего пользовательского интерфейса:

```
- (void)viewDidLoad{
  [super viewDidLoad];
  NSArray *segments = [[NSArray alloc] initWithObjects:
                       @"iPhone",
                       [UIImage imageNamed:@"iPad"],
                       @"iPod",
```

```
                        @"iMac",
                       ];

  self.mySegmentedControl = [[UISegmentedControl alloc]
                             initWithItems:segments];

  CGRect segmentedFrame = self.mySegmentedControl.frame;
  segmentedFrame.size.height = 128.0f;
  segmentedFrame.size.width = 300.0f;
  self.mySegmentedControl.frame = segmentedFrame;
  self.mySegmentedControl.center = self.view.center;

  [self.view addSubview:self.mySegmentedControl];
}
```

В данном примере файл iPad.png — это просто миниатюрное изображение «айпада», добавленное в наш проект.

В iOS 7 Apple отказалась от использования свойства `segmentedControlStyle` класса `UISegmentedControl`, поэтому теперь сегментированные элементы управления имеют всего один стиль, задаваемый по умолчанию. Мы больше не можем изменять этот стиль.

# 1.9. Представление видов и управление ими с помощью UIViewController

## Постановка задачи

Необходимо иметь возможность переключаться между видами в вашем приложении.

## Решение

Воспользуйтесь классом `UIViewController`.

## Обсуждение

Стратегия разработки для iOS, предложенная Apple, предполагает использование паттерна «модель — вид — контроллер» (MVC) и соответствующее разделение задач. Виды — это элементы, отображаемые для пользователя, а модель — это абстракция с данными, которыми управляет приложение. Контроллер — это перемычка, соединяющая модель и вид. Контроллер (в данном случае речь идет о контроллере вида) управляет отношениями между видом и моделью. Почему же этими отношениями не занимается вид? Ответ довольно прост: если бы мы возлагали

эти задачи на вид, код вида становился бы очень запутанным. Кроме того, такой подход тесно связывал бы виды с моделью, что не очень хорошо.

Контроллеры видов можно загружать из файлов XIB (для использования с конструктором интерфейсов) или просто создавать с помощью программирования. Сначала рассмотрим, как создать контроллер вида, *не пользуясь* файлом XIB.

Контроллеры видов удобно создавать в Xcode. Теперь, когда вы уже создали шаблон приложения с помощью шаблона Empty Application (Пустое приложение), выполните следующие шаги, чтобы создать новый контроллер вида для вашего приложения.

1. В Xcode перейдите в меню File (Файл) и там выберите New ▸ New File (Новый ▸ Новый файл).
2. В диалоговом окне New File (Новый файл) убедитесь, что слева выбраны категория iOS и подкатегория Cocoa Touch. Когда сделаете это, выберите класс `UIViewController` в правой части диалогового окна, а затем нажмите Next (Далее) (рис. 1.24).


**Рис. 1.24.** Подкласс нового контроллера вида

3. На следующем экране убедитесь, что в текстовом поле Subclass (Подкласс) указано `UIViewController`, а также что сняты флажки Targeted for iPad (Разработка для iPad) и With XIB for user interface (Использовать файл XIB для пользовательского интерфейса). Именно такая ситуация показана на рис. 1.25. Нажмите Next (Далее).

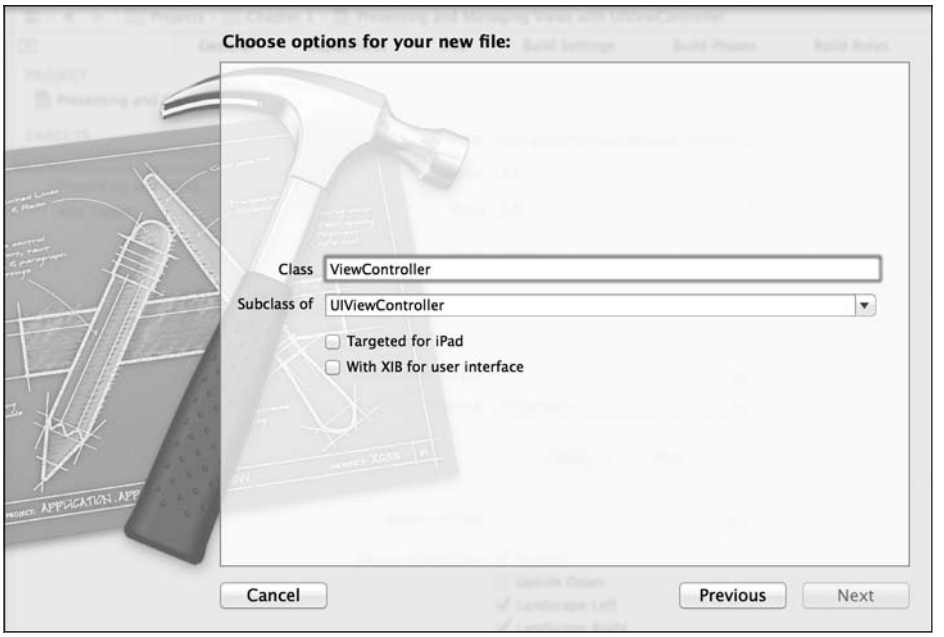


**Рис. 1.25.** Собственный контроллер вида, без использования класса XIB

4. На следующем экране (Save as (Сохранить как)) назовите файл контроллера вида `RootViewController` и нажмите Save (Сохранить) (рис. 1.26).

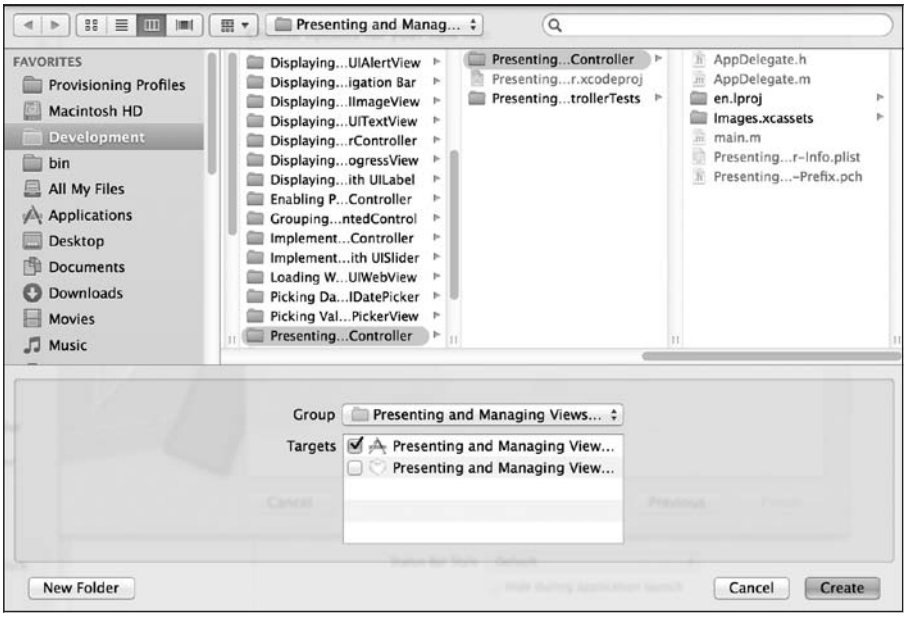


**Рис. 1.26.** Сохранение контроллера вида без использования файла XIB

5. Теперь найдем файл реализации (`.m`) делегата приложения, который обычно называется `AppDelegate.m`. В этом файле объявим свойство типа `ViewController`:

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

@interface AppDelegate ()
@property (nonatomic, strong) ViewController *viewController;
@end

@implementation AppDelegate

...
```

6. Найдем в файле реализации метод `application:didFinishLaunchingWithOptions:`, относящийся к делегату приложения, инстанцируем контроллер вида и добавим его в наше окно как корневой контроллер вида:

```
- (BOOL) application:(UIApplication *)application
  didFinishLaunchingWithOptions:(NSDictionary *)launchOptions{

    self.viewController = [[ViewController alloc] initWithNibName:nil
                                                            bundle:nil];

    self.window = [[UIWindow alloc]
                   initWithFrame:[[UIScreen mainScreen] bounds]];

    /* Делаем наш контроллер вида корневым контроллером вида */
    self.window.rootViewController = self.viewController;

    self.window.backgroundColor = [UIColor whiteColor];
    [self.window makeKeyAndVisible];
    return YES;
}
```

Теперь снова попробуем запустить приложение в эмуляторе. На экране увидим вид, имеющий ровный белый цвет. Поздравляю: вы только что создали контроллер вида, и теперь у вас есть доступ не только к контроллеру вида, но и к самому объекту этого вида.

Если при создании контроллера вида (см. рис. 1.25) установить флажок With XIB for user interface (Использовать файл XIB для пользовательского интерфейса), то Xcode также сгенерирует файл XIB. В таком случае вам придется загрузить контроллер вашего вида *из* этого файла XIB, передав в параметр `initWithNibName` метода `initWithNibName:bundle:` контроллера вида полное имя файла XIB:

```
- (BOOL) application:(UIApplication *)application
  didFinishLaunchingWithOptions:(NSDictionary *)launchOptions{

    self.viewController = [[ViewController alloc]
                           initWithNibName:@"ViewController"
```

```
                            bundle:nil];

    self.window = [[UIWindow alloc]
                   initWithFrame:[[UIScreen mainScreen] bounds]];

    /* Делаем наш контроллер вида корневым контроллером вида */
    self.window.rootViewController = self.viewController;

    self.window.backgroundColor = [UIColor whiteColor];
    [self.window makeKeyAndVisible];
    return YES;
}
```

Если вы все же создали файл XIB, подготавливая контроллер вашего вида, этот файл теперь можно выбрать в Xcode и смастерить пользовательский интерфейс в конструкторе интерфейсов.

## См. также

Раздел 1.0.

# 1.10. Предоставление возможностей совместного использования информации с применением UIActivityViewController

## Постановка задачи

Внутри вашего приложения вы хотите предоставить пользователям возможность обмениваться контентом с их друзьями. Для этого предполагается использовать интерфейс, подобный тому, что показан на рис. 1.27. В этом интерфейсе предоставляются различные возможности совместного использования информации, имеющиеся в iOS, — например, через Facebook и Twitter.

## Решение

Создайте экземпляр класса `UIActivityViewController` и реализуйте совместное использование контента в этом классе так, как рассказано в подразделе «Обсуждение» данного раздела.

Экземпляры класса UIActivityViewController на iPhone следует представлять модально, а на iPad — на вспомогательных экранах. Более подробно о вспомогательных экранах рассказано в разделе 1.29.

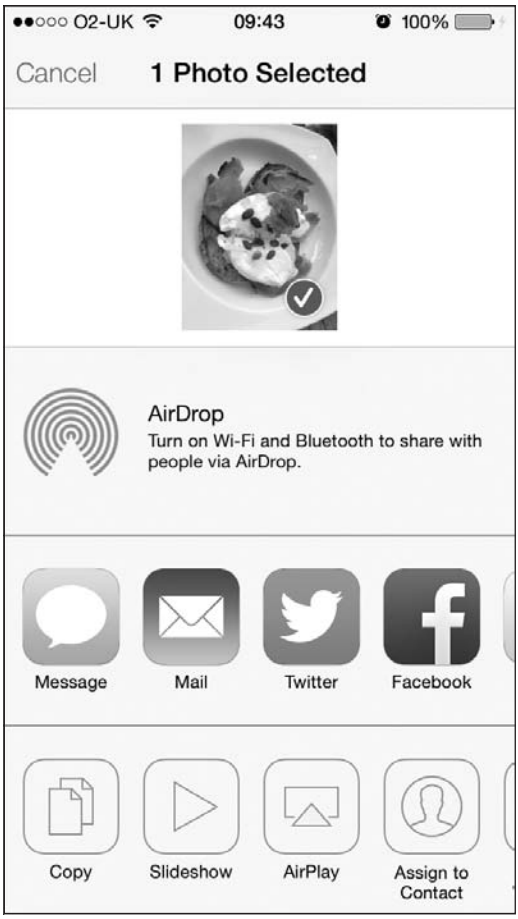


**Рис. 1.27.** Контроллер вида для обмена информацией, открытый на устройстве с iOS

## Обсуждение

В iOS существует масса возможностей совместного использования информации. Все они реализованы в ядре операционной системы. Например, такой неотъемлемой частью ядра сейчас является интеграция с Twitter и Facebook. Вы можете делиться практически любым контентом из этих сетей, находясь где угодно. Сторонние приложения наподобие того, которое собираемся написать мы, также могут использовать присущие iOS возможности совместного использования информации, не углубляясь в низкоуровневые детали сервисов и базовую организацию этих возможностей в iOS. Красота идеи заключается в том, что вам достаточно всего лишь указать, *чем* вы хотите поделиться, после чего iOS сама подберет возможности совместного использования, обеспечивающие обработку такой информации. Например, если вы хотите совместно использовать изображения и текст, то iOS предложит вам гораздо больше возможностей, чем если бы вы хотели поделиться аудиофайлом.

Совместное использование данных в iOS организовано очень просто. Для обеспечения такой работы вам всего лишь потребуется инстанцировать класс `UIActivityViewController` с помощью его метода-инициализатора `initWithActivityItems:applicationActivities:`. Вот какие параметры принимает этот метод:

- `initWithActivityItems` — массив элементов, которые предполагается совместно использовать. Это могут быть экземпляры `NSString`, `UIImage` или экземпляры любых других заказных классов, соответствующих протоколу `UIActivityItemSource`. Далее мы детально рассмотрим этот протокол;
- `applicationActivities` — массив экземпляров `UIActivity`, представляющих собой функции, поддерживаемые в вашем приложении. Например, здесь вы можете указать, может ли приложение организовать собственный механизм совместного использования изображений и строк. Пока мы не будем детально рассматривать этот параметр и просто передадим `nil` в качестве его значения. Так мы сообщаем iOS, что собираемся пользоваться только системными возможностями совместного использования.

Итак, допустим, что у нас есть текстовое поле, где пользователь может ввести текст, который затем будет использоваться совместно. Рядом с этим полем будет находиться кнопка Share (Поделиться). Когда пользователь нажимает кнопку Share, вы просто передаете текст, находящийся в текстовом поле, вашему экземпляру класса `UIActivityViewController`. Далее приведен соответствующий код. Мы пишем этот код для iPhone, поэтому представим контроллер вида с этой активностью как модальный контроллер вида.

Поскольку мы помещаем в нашем контроллере вида текстовое поле, нам необходимо обеспечить обработку его делегатных сообщений, в особенности тех, что поступают от метода `textFieldShouldReturn:` из протокола `UITextFieldDelegate`. Следовательно, мы собираемся выбрать контроллер вида в качестве делегата текстового поля. Кроме того, прикрепим к кнопке Share (Поделиться) метод действия. Когда эта кнопка будет нажата, нам потребуется убедиться, что в текстовом поле есть какая-то информация, которой можно поделиться. Если ее там не окажется, мы просто отобразим для пользователя окно с предупреждением, в котором сообщим, что не можем предоставить содержимое текстового поля для совместного использования. Если в текстовом поле окажется какой-либо текст, мы выведем на экран экземпляр класса `UIActivityViewController`.

Итак, начнем с файла реализации контроллера вида и определим компоненты пользовательского интерфейса:

```
@interface ViewController () <UITextFieldDelegate>
@property (nonatomic, strong) UITextField *textField;
@property (nonatomic, strong) UIButton *buttonShare;
@property (nonatomic, strong) UIActivityViewController *activityViewController;
@end

...
```

Затем напишем для контроллера вида два метода, каждый из которых будет способен создать один из компонентов пользовательского интерфейса и поместить

этот компонент в окно контроллера вида. Один метод будет создавать текстовое поле, а другой — кнопку рядом с этим полем:

```
- (void) createTextField{
  self.textField = [[UITextField alloc] initWithFrame:CGRectMake(20.0f,
                                                                 35.0f,
                                                                 280.0f,
                                                                 30.0f)];
  self.textField.translatesAutoresizingMaskIntoConstraints = NO;
  self.textField.borderStyle = UITextBorderStyleRoundedRect;
  self.textField.placeholder = @"Enter text to share...";
  self.textField.delegate = self;
  [self.view addSubview:self.textField];
}

- (void) createButton{
  self.buttonShare = [UIButton buttonWithType:UIButtonTypeRoundedRect];
  self.buttonShare.translatesAutoresizingMaskIntoConstraints = NO;
  self.buttonShare.frame = CGRectMake(20.0f, 80.0f, 280.0f, 44.0f);
  [self.buttonShare setTitle:@"Share" forState:UIControlStateNormal];

  [self.buttonShare addTarget:self
                       action:@selector(handleShare:)
             forControlEvents:UIControlEventTouchUpInside];

  [self.view addSubview:self.buttonShare];
}
```

Когда эта работа будет завершена, нам останется всего лишь вызвать два этих метода в методе `viewDidLoad` нашего контроллера вида. Таким образом мы правильно разместим компоненты пользовательского интерфейса в окне контроллера вида:

```
- (void)viewDidLoad{

  [super viewDidLoad];
  [self createTextField];
  [self createButton];

}
```

В методе `textFieldShouldReturn:` мы просто убираем с экрана клавиатуру, чтобы отказаться от активного состояния текстового поля. Это просто означает, что если пользователь редактировал текст в текстовом поле, а затем нажал клавишу Enter, то клавиатура должна исчезнуть с экрана. Не забывайте, что только что написанный метод `createTextField` задает наш контроллер вида в качестве делегата текстового поля. Поэтому потребуется реализовать упомянутый метод следующим образом:

```
- (BOOL) textFieldShouldReturn:(UITextField *)textField{
  [textField resignFirstResponder];
```

```
  return YES;
}
```

Последний, но немаловажный элемент — это метод-обработчик нашей кнопки. Как мы уже видели, метод `createButton` создает для нас кнопку и выбирает метод `handleShare:` для обработки действия-касания (нажатия) в рамках работы кнопки. Напишем этот метод:

```
- (void) handleShare:(id)paramSender{

  if ([self.textField.text length] == 0){
    NSString *message = @"Please enter a text and then press Share";
    UIAlertView *alertView = [[UIAlertView alloc] initWithTitle:nil
                                                        message:message
                                                       delegate:nil
                                              cancelButtonTitle:@"OK"
                                              otherButtonTitles:nil];
    [alertView show];
    return;
  }

  self.activityViewController = [[UIActivityViewController alloc]
                                 initWithActivityItems:@[self.textField.text]
                                 applicationActivities:nil];
  [self presentViewController:self.activityViewController
                     animated:YES
                   completion:^{
    /* Пока ничего не делаем */
  }];
}
```

Теперь, если запустить приложение, ввести в текстовое поле какой-либо текст, а затем нажать кнопку Share (Поделиться), мы получим результат, похожий на то, что изображено на рис. 1.28.

Вы можете выводить на экран параметры совместного использования уже вместе с контроллером вида. Метод `viewDidAppear` вашего контроллера вида будет вызываться, когда контроллер вида отобразится на экране и гарантированно окажется в иерархии видов вашего приложения. Это означает, что теперь вы сможете отобразить и другие виды поверх вашего контроллера вида.

Не пытайтесь представить контроллер вида для работы с функциями в методе viewDidLoad контроллера вида. На данном этапе подготовки приложения окно контроллера вашего вида еще не прикреплено к иерархии видов приложения, поэтому такая попытка ни к чему не приведет. Чтобы модальные виды работали, ваш вид должен быть частью такой иерархии. Поэтому необходимо представлять контроллер вида для обмена информацией в методе viewDidAppear контроллера вида.

## См. также

Раздел 1.29.

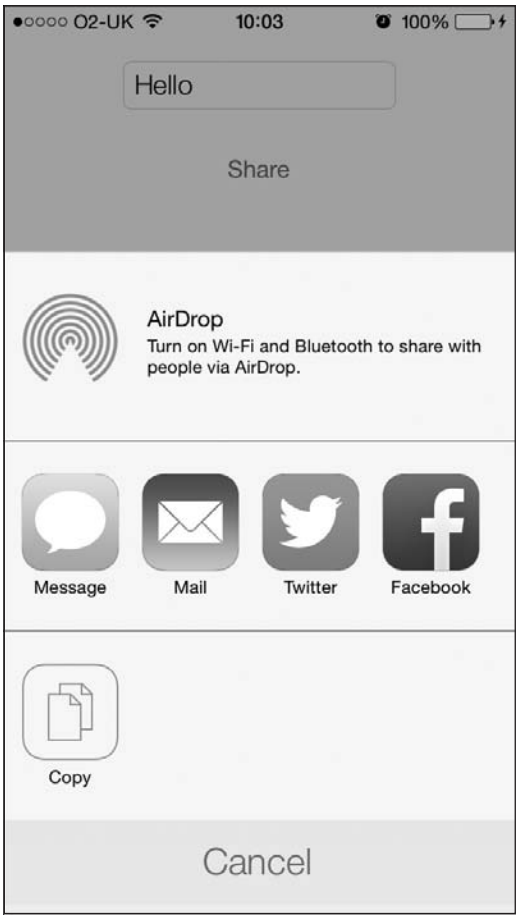


**Рис. 1.28.** Возможности совместного использования экземпляра строки, которым мы пытаемся поделиться

# 1.11. Предоставление специальных возможностей совместного использования данных с применением UIActivityViewController

## Постановка задачи

Вы хотите включить вашу программу в список тех приложений, которые способны обеспечивать в iOS совместную работу с данными и отображать эту программу в списке доступных функций, выстраиваемом в соответствующем контроллере вида (см. рис. 1.27).

Подобные возможности могут понадобиться вам, например, при работе с текстовым редактором. Когда пользователь нажимает кнопку Share (Поделиться), в контроллере вида с функцией должен появиться специальный элемент, в котором написано: Archive (Архивировать). Когда пользователь нажмет кнопку Archive (Архивировать), текст в редактируемой области вашего приложения будет передан специальной функции, а затем ваша функция сможет заархивировать этот текст в файловой системе на устройстве с iOS.

## Решение

Создайте класс типа `UIActivity`. Иными словами, произведите подкласс от этого класса и дайте новоиспеченному классу любое устраивающее вас имя. Экземпляры подклассов этого класса можно будет передавать методу-инициализатору `initWithActivityItems:applicationActivities:`, относящемуся к классу `UIActivityViewController`. Если эти экземпляры реализуют все необходимые методы класса `UIActivity`, то iOS отобразит их в контроллере вида с функцией.

## Обсуждение

Первый параметр метода `initWithActivityItems:applicationActivities:` принимает значения различных типов, в частности строки, числа, изображения и т. д. — фактически любые объекты. Если вы представите в параметре `initWithActivityItems` контроллер активности с массивом объектов произвольных типов, iOS просмотрит все доступные в системе функции — например, для работы с Facebook и Twitter — и предложит пользователю выбрать такую функцию, которая лучше всего отвечает его нуждам. После того как пользователь выберет функцию, iOS передаст *mun* объектов, находящихся в вашем массиве, в зарегистрированную системную функцию, выбранную пользователем. Затем такие функции смогут проверять тип объектов, которые вы собираетесь предоставлять в совместное пользование, и решать, может ли та или иная функция обработать такие объекты или нет. Функции передают такую информацию системе iOS посредством особого метода, реализуемого в их классах.

Итак, предположим, что мы хотим создать функцию, способную обратить любое количество переданных ей строк. Как вы помните, когда ваше приложение инициализирует контроллер вида с функцией с помощью метода `initWithActivityItems:applicationActivities:`, он может передать в первом параметре этого метода массив объектов произвольных типов. Поэтому если в функции планируется просмотреть все объекты, находящиеся в этом произвольном массиве, и если все они окажутся строками, то функция обратит их и отобразит все полученные строки в окне (виде) с предупреждением.

1. Произведите подкласс от `UIActivity` следующим образом:

```
#import <UIKit/UIKit.h>
@interface StringReverserActivity : UIActivity
@end
```

2. Поскольку мы собираемся выводить в нашей функции вид с предупреждением и отображать его для пользователя, когда нам будет передан массив строк, мы

должны гарантировать соответствие нашей функции протоколу UIAlertViewDelegate. Когда пользователь закроет окно с предупреждением, мы должны пометить нашу функцию как завершенную, вот так:

```
#import "StringReverserActivity.h"

@interface StringReverserActivity () <UIAlertViewDelegate>
@property (nonatomic, strong) NSArray *activityItems;
@end

@implementation StringReverserActivity

- (void) alertView:(UIAlertView *)alertView
  didDismissWithButtonIndex:(NSInteger)buttonIndex{
  [self activityDidFinish:YES];
}
```

3. Далее переопределим метод activityType нашей функции. Возвращаемое значение этого метода представляет собой объект типа NSString, являющийся уникальным идентификатором этой функции. Это значение не будет отображаться для пользователя — оно применяется только на уровне системы iOS для отслеживания идентификатора функции. Нет никаких особых значений, которые требовалось бы возвращать от этого метода, нет также никаких сопутствующих рекомендаций от Apple, но мы будем работать со строками в формате «обратное доменное имя», использовать идентификатор пакета приложения и прикреплять к нему имя нашего класса. Итак, если имеется идентификатор пакета com.pixolity.ios.cookbook.myapp и класс с именем StringReverserActivity, то мы возвратим от этого метода строку com.pixolity.ios.cookbook.myapp.StringReverserActivity, вот так:

```
- (NSString *) activityType{
  return [[NSBundle mainBundle].bundleIdentifier
          stringByAppendingFormat:@".%@", NSStringFromClass([self class])];
}
```

4. Следующий метод, который придется переопределить, называется activityTitle. В нем мы собираемся возвращать строку, которую будем отображать для пользователя в контроллере вида с функцией. Необходимо, чтобы эта строка получилась не слишком длинной и уместилась в нашем контроллере вида:

```
- (NSString *) activityTitle{
  return @"Reverse String";
}
```

5. Переходим к методу activityImage, который должен возвращать нам экземпляр UIImage — то самое изображение, что будет выводиться в контроллере вида с функцией. Обязательно предоставляйте по два варианта изображения — для сетчаточного дисплея и для обычного — как для iPad, так и для iPhone/iPod. Разрешение сетчаточного изображения для iPad должно составлять 110 × 110 пикселов, а для iPhone — 86 × 86 пикселов. Неудивительно, что, разделив эти значения на 2, получим ширину и высоту обычных изображений. В этом изображении iOS

использует только альфа-канал, поэтому убедитесь, что фон вашего изображения является прозрачным и что вы иллюстрируете его черным или белым цветом. Я уже создал изображение в разделе с ресурсами моего приложения и назвал его Reverse (Обратное). Вы можете ознакомиться с ним на рис. 1.29. А вот и код:

```
- (UIImage *) activityImage{
  return [UIImage imageNamed:@"Reverse"];
}
```

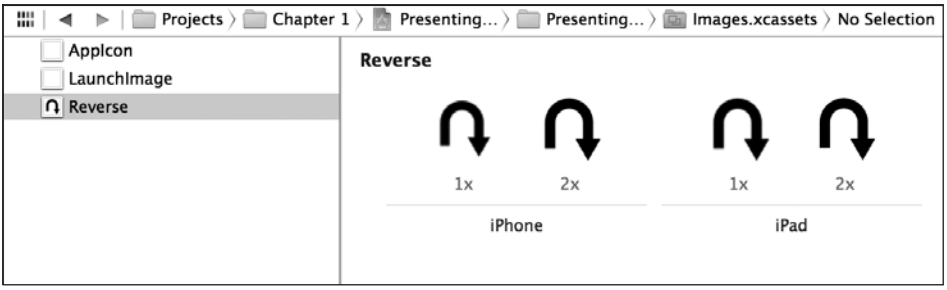


**Рис. 1.29.** В категории Ресурсы содержатся изображения для создаваемой специальной функции

6. Реализуем метод `canPerformWithActivityItems:` нашей функции. Параметр этого метода содержит массив, который будет задан, когда метод-инициализатор контроллера вида с функцией получит массив компонентов функции. Не забывайте, что тип каждого из объектов данного массива является произвольным. Возвращаемое значение данного метода является логическим и указывает, можем ли мы произвести такую функцию над каждым конкретным элементом массива. Например, наша функция может обратить любое количество данных ей строк. То есть если мы найдем в массиве одну строку, это будет нам на руку, поскольку мы будем точно знать, что впоследствии сможем обратить эту строку. Но если мы получим массив из 1000 объектов, ни один из которых не будет относиться к приемлемому для нас типу, мы отклоним такой запрос, вернув `NO` от данного метода:

```
- (BOOL) canPerformWithActivityItems:(NSArray *)activityItems{

  for (id object in activityItems){
    if ([object isKindOfClass:[NSString class]]){
      return YES;
    }
  }

  return NO;
}
```

7. Теперь реализуем метод `prepareWithActivityItems:` нашей функции, чей параметр относится к типу `NSArray`. Этот метод вызывается, если вы возвращаете `YES` от метода `canPerformWithActivityItems:`. Придется сохранить данный массив для

последующего использования. Но на самом деле можно сохранять не весь массив, а только часть его объектов — те, что относятся к интересующему вас типу. Например, строки:

```
- (void) prepareWithActivityItems:(NSArray *)activityItems{

  NSMutableArray *stringObjects = [[NSMutableArray alloc] init];
  for (id object in activityItems){
    if ([object isKindOfClass:[NSString class]]){
      [stringObjects addObject:object];
    }
  }

  self.activityItems = [stringObjects copy];
}
```

8. Последнее, но немаловажное: потребуется реализовать метод `performActivity` нашей функции, который вызывается, если iOS требует от нас произвести выбранные действия над списком ранее предоставленных произвольных объектов. В функции мы собираемся перебрать массив строковых объектов, извлеченных из массива с произвольными типами, обратить их все и отобразить для пользователя в окне с предупреждением:

```
- (NSString *) reverseOfString:(NSString *)paramString{

  NSMutableString *reversed = [[NSMutableString alloc]
  initWithCapacity:paramString.length];

  for (NSInteger counter = paramString.length - 1;
       counter >= 0;
       counter--){
        [reversed appendFormat:@"%c", [paramString characterAtIndex:counter]];
  }

  return [reversed copy];

}

- (void) performActivity{

  NSMutableString *reversedStrings = [[NSMutableString alloc] init];

  for (NSString *string in self.activityItems){
      [reversedStrings appendString:[self reverseOfString:string]];
      [reversedStrings appendString:@"\n"];
  }

  UIAlertView *alertView = [[UIAlertView alloc] initWithTitle:@"Reversed"
                                                      message:reversedStrings
                                                     delegate:self
                                            cancelButtonTitle:@"OK"
```

```
                                                    otherButtonTitles:nil];

    [alertView show];
  }
```

Итак, реализация класса нашей функции завершена. Перейдем к файлу реализации контроллера вида и отобразим контроллер вида функции в списке с нашей специальной функцией:

```
#import "ViewController.h"
#import "StringReverserActivity.h"

@implementation ViewController

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

  NSArray *itemsToShare = @[
                            @"Item 1",
                            @"Item 2",
                            @"Item 3",
                           ];

UIActivityViewController *activity =
[[UIActivityViewController alloc]
 initWithActivityItems:itemsToShare
 applicationActivities:@[[StringReverserActivity new]]];

[self presentViewController:activity animated:YES completion:nil];
}
@end
```

При первом запуске приложения на экране появится картинка, примерно такая, как на рис. 1.30.

Если теперь вы нажмете в этом списке элемент Reverse String (Обращенная строка), то увидите нечто похожее на рис. 1.31.

## См. также

Раздел 1.10.

# 1.12. Внедрение навигации с помощью UINavigationController

## Постановка задачи

Необходимо дать пользователю возможность переходить от одного контроллера вида к другому, сопровождая этот процесс плавной анимацией, интегрированной в программу.


**Рис. 1.30.** Специальная функция для обращения строк теперь находится в списке доступных функций


**Рис. 1.31.** Наша функция для обращения строк в действии

## Решение

Используйте экземпляр класса `UINavigationController`.

## Обсуждение

Если вам доводилось работать с iPhone, iPod touch или iPad, то вы, скорее всего, уже видели в действии навигационный инструмент управления. Например, если перейти в приложение Settings (Настройки) телефона, там можно выбрать команду Wallpaper (Обои) (рис. 1.32). В таком случае вы увидите, как основной экран программы Settings (Настройки) отодвигается влево, а на его место справа выходит экран Wallpaper (Обои). В этом и заключается самая интересная черта навигации iPhone. Вы можете *складывать* контроллеры видов в стек и *поднимать* их из стека.

Контроллер вида, в данный момент находящийся на верхней позиции стека, виден пользователю. Итак, только самый верхний контроллер вида показывается зрителю, а чтобы отобразить другой контроллер, нужно либо удалить с верхней позиции контроллер, видимый в настоящий момент, либо поместить на верхнюю позицию в стеке новый контроллер вида.


**Рис. 1.32.** Контроллер вида настроек, отодвигающий вид с обоями для экрана

Теперь добавим в новый проект навигационный контроллер. Но сначала нужно создать проект. Выполните шаги, описанные в разделе 1.9, чтобы создать пустое приложение с простым контроллером вида. Данный раздел — расширенная версия работы, выполненной в разделе 1.9. Начнем с файла реализации (`.m`) делегата нашего приложения:

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

@interface AppDelegate ()
```

```
@property (nonatomic, strong) UINavigationController *navigationController;
@end

@implementation AppDelegate
...
```

Теперь следует инициализировать навигационный контроллер, воспользовавшись его методом `initWithRootViewController:`, и передать корневой контроллер нашего вида как параметр этого метода. Далее мы зададим навигационный контроллер в в качестве корневого контроллера вида в нашем окне. Здесь главное — не запутаться. `UINavigationController` — это фактически подкласс `UIViewController`, а свойство `rootViewController`, относящееся к нашему окну, принимает любой объект типа `UIViewController`. Таким образом, если мы хотим сделать навигационный контроллер корневым контроллером нашего вида, мы просто должны задать его в качестве корневого контроллера:

```
- (BOOL) application:(UIApplication *)application
  didFinishLaunchingWithOptions:(NSDictionary *)launchOptions{

    FirstViewController *viewController = [[FirstViewController alloc]
                                           initWithNibName:nil
                                           bundle:nil];

    self.navigationController = [[UINavigationController alloc]
                                 initWithRootViewController:viewController];

    self.window = [[UIWindow alloc]
                   initWithFrame:[[UIScreen mainScreen] bounds]];

    self.window.rootViewController = self.navigationController;

    self.window.backgroundColor = [UIColor whiteColor];
    [self.window makeKeyAndVisible];
    return YES;
}
```

После этого запустим приложение в эмуляторе (рис. 1.33).

Файл реализации корневого контроллера вида создает кнопку в центре экрана (как показано на рис. 1.33). Чуть позже мы изучим этот файл реализации.

На рис. 1.33 мы в первую очередь замечаем полосу в верхней части экрана. Теперь экран уже не чисто-белый. Что это за новый виджет? Это навигационная панель. Мы будем активно пользоваться ею при навигации, например разместим на ней кнопки и сделаем кое-что еще. Кроме того, на этой панели удобно отображать заголовок. Каждый контроллер вида сам для себя указывает заголовок, а навигационный контроллер будет автоматически отображать заголовок того контроллера вида, который окажется на верхней позиции в стеке.

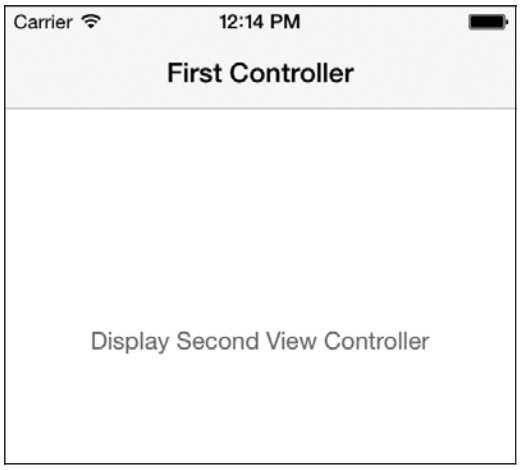


**Рис. 1.33.** Пустой контроллер вида, отображаемый внутри навигационного контроллера

Переходим к файлу реализации корневого контроллера нашего вида в методе `viewDidLoad`. В качестве свойства контроллера вида укажем `First Controller`. Здесь же создадим кнопку. Когда пользователь нажмет эту кнопку, мы отобразим на экране второй контроллер вида:

```
#import "FirstViewController.h"
#import "SecondViewController.h"

@interface FirstViewController ()
@property (nonatomic, strong) UIButton *displaySecondViewController;
@end

@implementation FirstViewController

- (void) performDisplaySecondViewController:(id)paramSender{
  SecondViewController *secondController = [[SecondViewController alloc]
                                            initWithNibName:nil
                                            bundle:NULL];
  [self.navigationController pushViewController:secondController
                                       animated:YES];
}

- (void)viewDidLoad{
  [super viewDidLoad];
  self.title = @"First Controller";

  self.displaySecondViewController = [UIButton
                                      buttonWithType:UIButtonTypeSystem];

  [self.displaySecondViewController
  setTitle:@"Display Second View Controller"
```

```
  forState:UIControlStateNormal];

  [self.displaySecondViewController sizeToFit];
  self.displaySecondViewController.center = self.view.center;
  [self.displaySecondViewController
  addTarget:self
  action:@selector(performDisplaySecondViewController:)
  forControlEvents:UIControlEventTouchUpInside];

  [self.view addSubview:self.displaySecondViewController];
}

@end
```

А теперь создадим второй контроллер вида, уже без файла XIB, и назовем его `SecondViewController`. Проделайте тот же процесс, что был показан в разделе 1.9. Когда создадите этот контроллер вида, назовите его `Second Controller`:

```
#import "SecondViewController.h"

@implementation SecondViewController

- (void)viewDidLoad{
  [super viewDidLoad];
  self.title = @"Second Controller";
}
```

Теперь мы собираемся *всплыть* из второго контроллера вида обратно в первый контроллер вида через 5 секунд после того, как первый контроллер вида окажется на экране. Для этого используем метод `performSelector:withObject:afterDelay:` объекта `NSObject`, чтобы вызвать новый метод `goBack`. Второй метод будет вызван через 5 секунд после того, как контроллер первого вида успешно отобразит на экране этот первый вид. В методе `goBack` просто используем свойство `navigationController` контроллера вида (а оно встроено в `UIViewController`, и нам самим не приходится его писать), чтобы вернуться к экземпляру `FirstViewController`. Для этого воспользуемся методом `popViewControllerAnimated:` навигационного контроллера, который принимает в качестве параметра логическое значение. Если этот параметр имеет значение `YES`, то переход к предыдущему контроллеру вида будет анимироваться, если `NO` — не будет. В результате мы увидим примерно такую картинку, как на рис. 1.34.

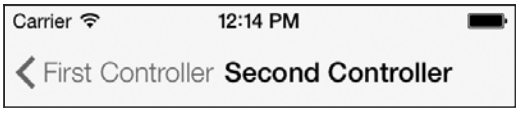


**Рис. 1.34.** Контроллер вида размещается поверх другого контроллера вида

```
#import "SecondViewController.h"

@implementation SecondViewController
```

```
- (void)viewDidLoad{
  [super viewDidLoad];
  self.title = @"Second Controller";
}

- (void) goBack{
  [self.navigationController popViewControllerAnimated:YES];
}

- (void) viewDidAppear:(BOOL)paramAnimated{
  [super viewDidAppear:paramAnimated];
  [self performSelector:@selector(goBack)
  withObject:nil
  afterDelay:5.0f];

}

@end
```

Как видите, на навигационной панели отображается заголовок вида, занимающего верхнюю позицию в стеке, и даже имеется кнопка Назад, которая позволяет пользователю вернуться к контроллеру предыдущего вида. В стек вы можете поместить столько контроллеров видов, сколько хотите, и навигационный контроллер сработает так, чтобы на навигационной панели отображались кнопки Назад, работающие правильно и позволяющие пользователю пролистать назад весь графический интерфейс приложения, до самого первого вида.

Итак, если вы теперь откроете приложение в эмуляторе и подождете 5 секунд после того, как отобразится контроллер первого вида, то увидите, что по истечении этого времени на экране автоматически появится контроллер второго вида. Подождите еще 5 секунд — и второй контроллер вида автоматически уйдет с экрана, освободив место первому.

## См. также

Раздел 1.9.

# 1.13. Управление массивом контроллеров видов, относящихся к навигационному контроллеру

## Постановка задачи

Требуется возможность непосредственно управлять массивом контроллеров видов, связанных с конкретным навигационным контроллером.

## Решение

Воспользуйтесь свойством `viewControllers` из класса `UINavigationController` для доступа к массиву контроллеров видов, связанных с навигационным контроллером, а также для изменения этого массива:

```
- (void) goBack{
  /* Получаем актуальный массив контроллеров видов. */
  NSArray *currentControllers = self.navigationController.viewControllers;

  /* Создаем на основе этого массива изменяемый массив. */
  NSMutableArray *newControllers = [NSMutableArray
                                    arrayWithArray:currentControllers];

  /* Удаляем последний объект из массива. */
  [newControllers removeLastObject];

  /* Присваиваем этот массив навигационному контроллеру. */
  self.navigationController.viewControllers = newControllers
}
```

Этот метод можно вызвать внутри любого контроллера вида, чтобы поднять последний контроллер вида из иерархии навигационного контроллера, связанного с контроллером вида, который отображается в настоящий момент.

## Обсуждение

Экземпляр класса `UINavigationController` содержит массив объектов `UIViewController`. Получив этот массив, вы можете оперировать им как угодно. Например, можно удалить контроллер вида из произвольного места в массиве.

Если мы напрямую управляем контроллерами видов, связанными с навигационным контроллером, то есть путем присвоения массива свойству `viewControllers` навигационного контроллера, то весь процесс будет протекать без явного перехода между контроллерами и без анимации. Если вы хотите, чтобы эти действия анимировались, используйте метод `setViewControllers:animated:`, относящийся к классу `UINavigationController`, как показано в следующем фрагменте кода:

```
- (void) goBack{
  /* Получаем актуальный массив контроллеров видов. */
  NSArray *currentControllers = self.navigationController.viewControllers;

  /* Создаем на основе этого массива изменяемый массив. */
  NSMutableArray *newControllers = [NSMutableArray
                                   arrayWithArray:currentControllers];

  /* Удаляем последний объект из массива. */
  [newControllers removeLastObject];

  /* Присваиваем этот массив навигационному контроллеру. */
```

```
  [self.navigationController setViewControllers:newControllers
                                       animated:YES];
}
```

# 1.14. Демонстрация изображения на навигационной панели

## Постановка задачи

В качестве заголовка контроллера вида, ассоциированного в данный момент с навигационным контроллером, требуется отобразить не текст, а изображение.

## Решение

Воспользуйтесь свойством `titleView` навигационного элемента контроллера вида:

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

  /* Создаем вид с изображением, заменяя им вид с заголовком. */
  UIImageView *imageView =
  [[UIImageView alloc]
   initWithFrame:CGRectMake(0.0f, 0.0f, 100.0f, 40.0f)];

  imageView.contentMode = UIViewContentModeScaleAspectFit;

  /* Загружаем изображение. Внимание! Оно будет кэшироваться. */
  UIImage *image = [UIImage imageNamed@"Logo"];

  /* Задаем картинку для вида с изображением. */
  [imageView setImage:image];

  /* Задаем вид с заголовком. */
  self.navigationItem.titleView = imageView;

}
```

Предыдущий код должен выполняться в контроллере вида, находящемся внутри навигационного контроллера.

Я уже загрузил изображение в группу ресурсов моего проекта и назвал это изображение Logo. Как только вы запустите это приложение с приведенным фрагментом кода, увидите результат, напоминающий рис. 1.35.


**Рис. 1.35.** Вид с изображением на нашей навигационной панели

## Обсуждение

Навигационный элемент каждого конкретного контроллера вида может отображать два различных вида контента в той области контроллера вида, которой этот элемент присвоен:

- обычный текст;
- вид.

Если вы собираетесь работать с текстом, можете использовать свойство `title` навигационного элемента. Тем не менее, если вам требуется более полный контроль над заголовком или вы просто хотите вывести над навигационной панелью изображение или любой другой вид, можете использовать свойство `titleView` навигационного элемента контроллера вида. Ему можно присваивать любой объект, являющийся подклассом класса `UIView`. В примере мы создали вид для изображения, а затем присвоили ему изображение. Потом вывели это изображение в качестве заголовка вида, в настоящий момент находящегося на навигационном контроллере.

Свойство `titleView` навигационной панели — это самый обычный вид, но Apple рекомендует, чтобы его высота не превышала 128 точек. Поэтому считайте его изображением. Если бы вы загружали изображение, имеющее высоту 128 *пикселов*, то *на сетчатом дисплее это соответствовало бы 64 точкам* и все было бы нормально. Но если бы вы загружали изображение высотой 300 пикселов на сетчатом дисплее, то по высоте оно заняло бы 150 точек, то есть заметно превысило бы те 128 точек, которые Apple рекомендует для видов, расположенных в строке заголовка. Для исправления этой ситуации необходимо гарантировать, что вид в строке заголовка по высоте ни в коем случае не окажется больше 128 точек, а также задать для контента режим заполнения вида целиком, а не подгонки вида под содержимое. Для этого можно установить свойство `contentMode` вашей строки заголовка в `UIViewContentModeScaleAspectFit`.

# 1.15. Добавление кнопок на навигационные панели с помощью UIBsrButtonItem

## Постановка задачи

Необходимо добавить кнопки на навигационную панель.

## Решение

Используйте класс `UIBarButtonItem`.

## Обсуждение

На навигационной панели могут содержаться различные элементы. Кнопки часто отображаются в ее левой и правой частях. Такие кнопки относятся к классу `UIBarButtonItem` и могут принимать самые разнообразные формы и очертания. Рассмотрим пример, показанный на рис. 1.36.

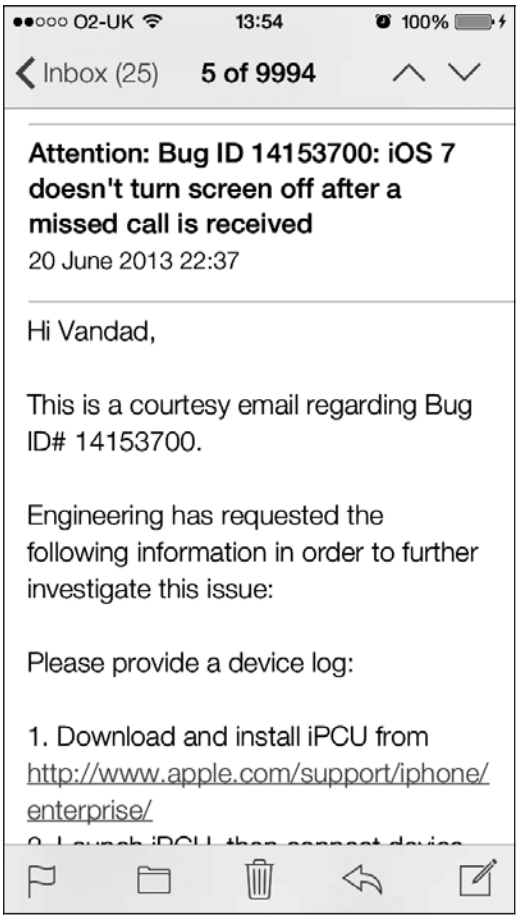


**Рис. 1.36.** Различные кнопки, отображаемые на навигационной панели

Навигационные панели относятся к классу `UINavigationBar`, их можно создавать когда угодно и добавлять к любому виду. Итак, просто рассмотрим разные кнопки (с разными очертаниями), добавленные на навигационные панели на рис. 1.36. На кнопках, размещенных справа сверху, видим стрелки, которые направлены вверх и вниз. На кнопке, находящейся слева вверху, имеется стрелка, указывающая влево. Кнопки, расположенные на нижней навигационной панели, имеют разные очертания. В этом разделе мы рассмотрим, как создаются некоторые из таких кнопок.

Работая с данным разделом, выполните шаги, перечисленные в подразделе «Создание и запуск вашего первого приложения для iOS» раздела 1.0 данной главы и создайте пустое приложение. Потом проделайте шаги, описанные в разделе 1.12, и добавьте в делегат вашего приложения навигационный контроллер.

Чтобы создать кнопку для навигационной панели, необходимо сделать следующее.

1. Создать экземпляр класса `UIBarButtonItem`.
2. Добавить получившуюся кнопку на навигационную панель, воспользовавшись свойством `navigationItem`, относящимся к контроллеру вида. Свойство `navigationItem` позволяет взаимодействовать с навигационной панелью. Само это свойство может принимать еще два свойства: `rightBarButtonItem` и `leftBarButtonItem`. Оба они относятся к типу `UIBarButtonItem`.

Теперь рассмотрим пример, в котором добавим кнопку в правую часть нашей навигационной панели. На этой кнопке будет написано Add (Добавить):

```
- (void) performAdd:(id)paramSender{
  NSLog(@"Action method got called.");
}

- (void)viewDidLoad{
  [super viewDidLoad];

  self.title = @"First Controller";

  self.navigationItem.rightBarButtonItem =
  [[UIBarButtonItem alloc] initWithTitle:@"Add"
                                   style:UIBarButtonItemStylePlain
                                  target:self
                                  action:@selector(performAdd:)];
}
```

Если сейчас запустить приложение, появится картинка, примерно как на рис. 1.37.

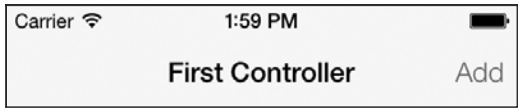


**Рис. 1.37.** Навигационная кнопка, добавленная на навигационную панель

Пока все просто. Но если вы регулярно пользуетесь iOS, то, вероятно, заметили, что в системных приложениях iOS применяется готовая конфигурация и кнопка Add (Добавить) там выглядит иначе. На рис. 1.38 показан пример из раздела Alarm (Будильник) приложения Clock (Часы) для iPhone. Обратите внимание на кнопку + в верхней правой части навигационной панели.

Оказывается, в SDK iOS можно создавать *системные* кнопки. Это делается с помощью метода-инициализатора `nitWithBarButtonSystemItem:target:action:`, относящегося к классу `UIBarButtonItem`:

```
- (void) performAdd:(id)paramSender{
  NSLog(@"Action method got called.");
```

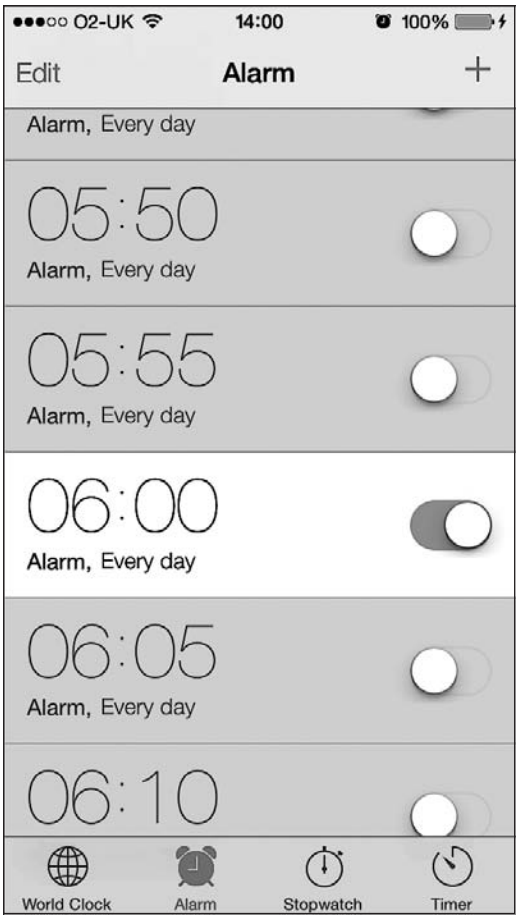


**Рис. 1.38.** Правильный способ создания кнопки Add (Добавить)

```
}

- (void)viewDidLoad{
  [super viewDidLoad];
  self.title = @"First Controller";

  self.navigationItem.rightBarButtonItem =
  [[UIBarButtonItem alloc]
   initWithBarButtonSystemItem:UIBarButtonSystemItemAdd
                        target:self
                        action:@selector(performAdd:)];
}
```

В результате получится именно то, чего мы добивались (рис. 1.39).

Первый параметр метода-инициализатора `initWithBarButtonSystemItem:target:action:`, относящегося к навигационной кнопке, может принимать в качестве параметров любые значения из перечня `UIBarButtonSystemItem`:

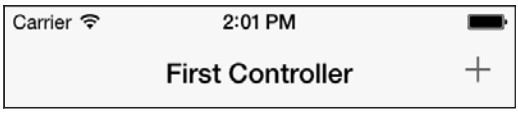


**Рис. 1.39.** Системная кнопка Add (Добавить)

```
typedef NS_ENUM(NSInteger, UIBarButtonSystemItem) {
  UIBarButtonSystemItemDone,
  UIBarButtonSystemItemCancel,
  UIBarButtonSystemItemEdit,
  UIBarButtonSystemItemSave,
  UIBarButtonSystemItemAdd,
  UIBarButtonSystemItemFlexibleSpace,
  UIBarButtonSystemItemFixedSpace,
  UIBarButtonSystemItemCompose,
  UIBarButtonSystemItemReply,
  UIBarButtonSystemItemAction,
  UIBarButtonSystemItemOrganize,
  UIBarButtonSystemItemBookmarks,
  UIBarButtonSystemItemSearch,
  UIBarButtonSystemItemRefresh,
  UIBarButtonSystemItemStop,
  UIBarButtonSystemItemCamera,
  UIBarButtonSystemItemTrash,
  UIBarButtonSystemItemPlay,
  UIBarButtonSystemItemPause,
  UIBarButtonSystemItemRewind,
  UIBarButtonSystemItemFastForward,
#if __IPHONE_3_0 <= __IPHONE_OS_VERSION_MAX_ALLOWED
  UIBarButtonSystemItemUndo,
  UIBarButtonSystemItemRedo,
#endif
#if __IPHONE_4_0 <= __IPHONE_OS_VERSION_MAX_ALLOWED
  UIBarButtonSystemItemPageCurl,
#endif
};
```

Один из самых интересных инициализаторов из класса `UIBarButtonItem` — метод `initWithCustomView:`. В качестве параметра этот метод может принимать любой вид, то есть мы даже можем добавить на навигационную панель в качестве навигационной кнопки `UISwitch` (см. раздел 1.2). Это будет выглядеть не очень красиво, но мы просто попробуем:

```
- (void) switchIsChanged:(UISwitch *)paramSender{
  if ([paramSender isOn]){
    NSLog(@"Switch is on.");
  } else {
    NSLog(@"Switch is off.");
  }
}
```

```
- (void)viewDidLoad{
  [super viewDidLoad];
  self.view.backgroundColor = [UIColor whiteColor];
  self.title = @"First Controller";

  UISwitch *simpleSwitch = [[UISwitch alloc] init];
  simpleSwitch.on = YES;
  [simpleSwitch addTarget:self
                   action:@selector(switchIsChanged:)
         forControlEvents:UIControlEventValueChanged];

  self.navigationItem.rightBarButtonItem =
  [[UIBarButtonItem alloc] initWithCustomView:simpleSwitch];
}
```

Вот что получается (рис. 1.40).

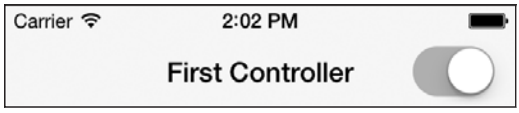


**Рис. 1.40.** Переключатель, добавленный на навигационную панель

На навигационной панели можно создавать очень и очень занятные кнопки. Просто взгляните, что делает Apple со стрелками, направленными вверх и вниз, расположенными в правом верхнем углу на рис. 1.36. А почему бы нам тоже так не сделать? Впечатление такое, как будто в кнопку встроен сегментированный элемент управления (см. раздел 1.8). Итак, нам нужно создать такой элемент управления с двумя сегментами, добавить его на навигационную кнопку и, наконец, поставить эту кнопку на навигационную панель. Начнем:

```
- (void) segmentedControlTapped:(UISegmentedControl *)paramSender{
  switch (paramSender.selectedSegmentIndex){
    case 0:{
      NSLog(@"Up");
      break;
    }
    case 1:{
      NSLog(@"Down");
      break;
    }
  }

}

- (void)viewDidLoad{
  [super viewDidLoad];

  self.title = @"First Controller";
```

```
  NSArray *items = @[
                     @"Up",
                     @"Down"
                    ];

  UISegmentedControl *segmentedControl = [[UISegmentedControl alloc]
                                          initWithItems:items];

  segmentedControl.momentary = YES;

  [segmentedControl addTarget:self
                       action:@selector(segmentedControlTapped:)
             forControlEvents:UIControlEventValueChanged];

  self.navigationItem.rightBarButtonItem =
  [[UIBarButtonItem alloc] initWithCustomView:segmentedControl];
}
```

На рис. 1.41 показано, что должно получиться в итоге.

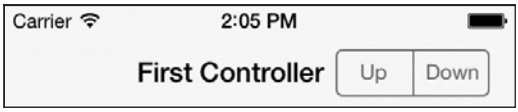


**Рис. 1.41.** Сегментированный элемент управления, встроенный в навигационную кнопку

Элемент `navigationItem` любого контроллера вида имеет еще два замечательных метода:

- `setRightBarButtonItem:animated:` — задает правую кнопку навигационной панели;
- `setLeftBarButtonItem:animated:` — определяет левую кнопку навигационной панели.

Оба метода позволяют указывать, хотите ли вы анимировать кнопку. Задайте значение `YES` для параметра `animated`, если анимация нужна:

```
UIBarButtonItem *rightBarButton =
[[UIBarButtonItem alloc] initWithCustomView:segmentedControl];

[self.navigationItem setRightBarButtonItem:rightBarButton
                                  animated:YES];
```

## См. также

Подраздел «Создание и запуск вашего первого приложения для iOS» раздела 1.0 данной главы. Разделы 1.2, 1.8, 1.12.

# 1.16. Представление контроллеров, управляющих несколькими видами, с помощью UITabBarController

## Постановка задачи

Необходимо дать пользователям возможность переключаться из одного раздела вашего приложения в другой, причем делать это просто.

## Решение

Используйте класс `UITabBarController`.

## Обсуждение

Если вы пользуетесь iPhone как будильником, то, разумеется, замечали на экране панель вкладок. Взгляните на рис. 1.38. В нижней части экрана расположены значки, которые называются World Clock (Мировое время), Alarm (Будильник), Stopwatch (Секундомер) и Timer (Таймер). Вся черная полоса в нижней части экрана — это панель вкладок, а вышеупомянутые ярлыки — ее элементы.

Панель вкладок — это контейнерный контроллер. Это значит, что мы создаем экземпляры `UITabBarController` и добавляем их в окно нашего приложения. Для каждого элемента панели вкладок мы добавляем на эту панель навигационный контроллер или контроллер вида. Эти элементы будут отображаться как вкладки на панели. Контроллер панели вкладок содержит панель вкладок типа `UITabBar`. Мы не создаем этот объект вручную — мы создаем контроллер панели вкладок, а уже он создает для нас такой объект. Проще говоря, считайте, что мы инстанцируем контроллер панели вкладок, а потом задаем контроллеры видов для этой панели. Данные контроллеры видов будут относиться к типу `UIViewController` или `UINavigationController`, если мы собираемся создать по контроллеру для каждого элемента панели вкладки (они же — контроллеры видов, задаваемые для контроллера панели вкладок). Навигационные контроллеры относятся к типу `UINavigationController` и являются подклассами от `UIViewController`. Следовательно, навигационный контроллер — это контроллер вида, но контроллеры видов, относящиеся к типу `UIViewController`, не являются навигационными контроллерами.

Итак, предположим, что у нас есть два контроллера видов. Классы этих контроллеров называются `FirstViewController` и `SecondViewController`:

```
- (BOOL) application:(UIApplication *)application
  didFinishLaunchingWithOptions:(NSDictionary *)launchOptions{

    self.window = [[UIWindow alloc] initWithFrame:
                   [[UIScreen mainScreen] bounds]];

    [self.window makeKeyAndVisible];
```

```
    FirstViewController *firstViewController = [[FirstViewController alloc]
                                                initWithNibName:nil
                                                        bundle:NULL];

    SecondViewController *secondViewController = [[SecondViewController alloc]
                                                  initWithNibName:nil
                                                          bundle:NULL];

    UITabBarController *tabBarController = [[UITabBarController alloc] init];
    [tabBarController setViewControllers:@[firstViewController,
                                           secondViewController
                                          ]];

    self.window.rootViewController = tabBarController;

    return YES;
}
```

Когда панель вкладок отобразится на экране, ее элементы будут расположены именно так, как показано на рис. 1.38. Имя каждого из этих элементов основывается на названии того контроллера вида, который соответствует конкретному элементу. Определим заголовки для обоих контроллеров наших видов.

Когда загружается панель вкладок, вместе с ней загружается контроллер вида первого входящего в нее элемента. Все остальные контроллеры видов инициализируются, но их виды не загружаются. Это означает, что любой код, который вы напишете во viewDidLoad второго контроллера вида, не выполнится до тех пор, пока пользователь не нажмет второй элемент этой панели в первый раз. Поэтому если вы присвоите заголовок панели контроллеру второго вида в его viewDidLoad и запустите приложение, то обнаружите, что заголовок панели вкладок по-прежнему пуст.

Первый контроллер вида мы назовем `First`:

```
#import "FirstViewController.h"

@implementation FirstViewController

- (id)initWithNibName:(NSString *)nibNameOrNil
               bundle:(NSBundle *)nibBundleOrNil{

  self = [super initWithNibName:nibNameOrNil
                         bundle:nibBundleOrNil];
  if (self != nil) {
      self.title = @"First";
  }
  return self;

}

- (void)viewDidLoad{
```

```
  [super viewDidLoad];
  self.view.backgroundColor = [UIColor whiteColor];
}
```

А второй контроллер вида будет называться Second:

```
#import "SecondViewController.h"

@implementation SecondViewController

- (id)initWithNibName:(NSString *)nibNameOrNil
               bundle:(NSBundle *)nibBundleOrNil{

  self = [super initWithNibName:nibNameOrNil
                         bundle:nibBundleOrNil];
  if (self != nil) {
      self.title = @"Second";
  }
  return self;

}

- (void)viewDidLoad{
  [super viewDidLoad];
  self.view.backgroundColor = [UIColor whiteColor];

}
```

Теперь запустим приложение и посмотрим, что получилось (рис. 1.42).

Как видите, у контроллеров видов нет навигационной панели. Что делать? Все просто. Как вы помните, UINavigationController — это подкласс UIViewController. Итак, мы можем добавлять экземпляры навигационных контроллеров на панель вкладок, а внутрь каждого навигационного контроллера загрузить контроллер вида. Чего же мы ждем?

```
- (BOOL) application:(UIApplication *)application
  didFinishLaunchingWithOptions:(NSDictionary *)launchOptions{

    // Точка переопределения для специальной настройки,
    // выполняемой после запуска приложения.
    self.window = [[UIWindow alloc] initWithFrame:
                   [[UIScreen mainScreen] bounds]];

    [self.window makeKeyAndVisible];

   FirstViewController *firstViewController = [[FirstViewController alloc]
                                               initWithNibName:nil
                                               bundle:NULL];

   UINavigationController *firstNavigationController =
     [[UINavigationController alloc]
      initWithRootViewController:firstViewController];
```

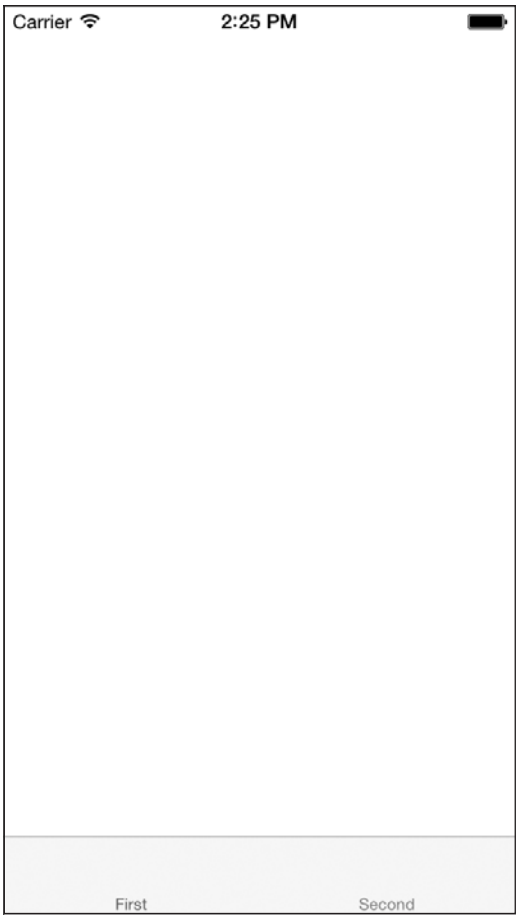


**Рис. 1.42.** Очень простая панель вкладок, на которой находятся два контроллера вида

```
SecondViewController *secondViewController = [[SecondViewController alloc]
                                               initWithNibName:nil
                                               bundle:NULL];

UINavigationController *secondNavigationController =
  [[UINavigationController alloc]
   initWithRootViewController:secondViewController];

UITabBarController *tabBarController = [[UITabBarController alloc] init];

[tabBarController setViewControllers:
  @[firstNavigationController, secondNavigationController]];

self.window.rootViewController = tabBarController;
```

```
    return YES;
}
```

Что получается? Именно то, что мы хотели (рис. 1.43).

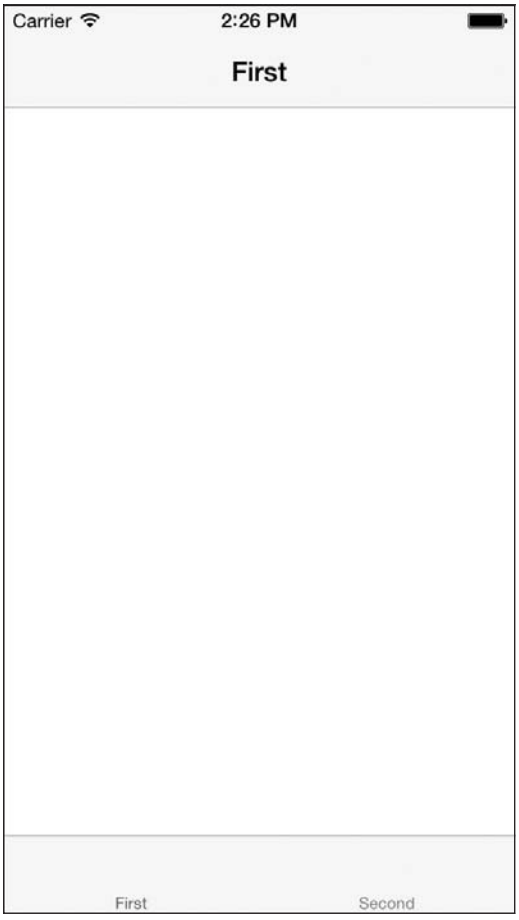


**Рис. 1.43.** Панель вкладок, на которой контроллеры видов находятся внутри навигационных контроллеров

Как было показано на рис. 1.38, каждый элемент панели вкладок может содержать текст или изображение. Мы узнали, что, пользуясь свойством `title` контроллера вида, можно задавать такой текст. А что насчет изображения? Оказывается, у каждого контроллера вида есть и свойство `tabItem`. Это свойство соответствует той вкладке, которая находится в актуальном контроллере вида. Вы можете пользоваться этим свойством, чтобы задавать изображение для вкладки. Изображение для вкладки задается через ее свойство `image`. Я уже сделал два изображения — прямоугольник и кружок, а теперь выведу их как изображения для вкладок,

соответствующих каждому из моих контроллеров видов. Вот код для первого контроллера вида:

```
- (id)initWithNibName:(NSString *)nibNameOrNil
               bundle:(NSBundle *)nibBundleOrNil{

  self = [super initWithNibName:nibNameOrNil
  bundle:nibBundleOrNil];
  if (self != nil) {
    self.title = @"First";
    self.tabBarItem.image = [UIImage imageNamed:@"FirstTab"];
  }
  return self;

}
- (void)viewDidLoad{
  [super viewDidLoad];
  self.view.backgroundColor = [UIColor whiteColor];
}
```

А вот код для второго контроллера:

```
- (id)initWithNibName:(NSString *)nibNameOrNil
  bundle:(NSBundle *)nibBundleOrNil{

    self = [super initWithNibName:nibNameOrNil
    bundle:nibBundleOrNil];
    if (self != nil) {
        self.title = @"Second";
        self.tabBarItem.image = [UIImage imageNamed:@"SecondTab"];
    }
    return self;

}

- (void)viewDidLoad{
  [super viewDidLoad];
  self.view.backgroundColor = [UIColor whiteColor];
}
```

Запустив приложение в эмуляторе, увидим такую картинку, как на рис. 1.44.

# 1.17. Отображение статического текста с помощью UILabel

## Постановка задачи

Необходимо отображать для пользователя текст. Кроме того, вы хотели бы управлять шрифтом и цветом этого текста.

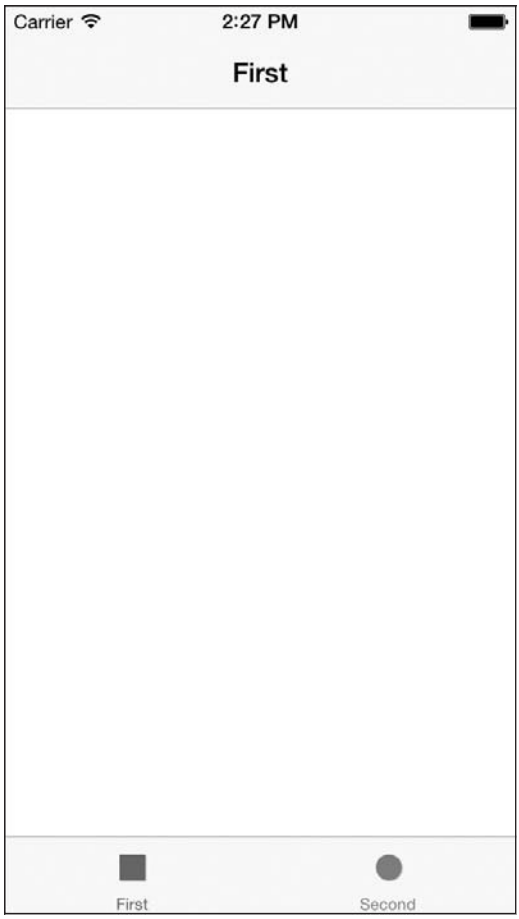


**Рис. 1.44.** Элементы панели вкладок с изображениями

Статическим называется такой текст, который пользователь не может напрямую изменять во время исполнения.

## Решение

Используйте класс `UILabel`.

## Обсуждение

Подписи (Labels) встречаются в iOS повсюду. Они используются практически в любых приложениях, за исключением игр, для отображения содержимого которых обычно применяется OpenGL ES, а не основные фреймворки отрисовки, входящие в состав iOS. На рис. 1.45 показаны несколько подписей, имеющихся в приложении Settings (Настройки) для iPhone.

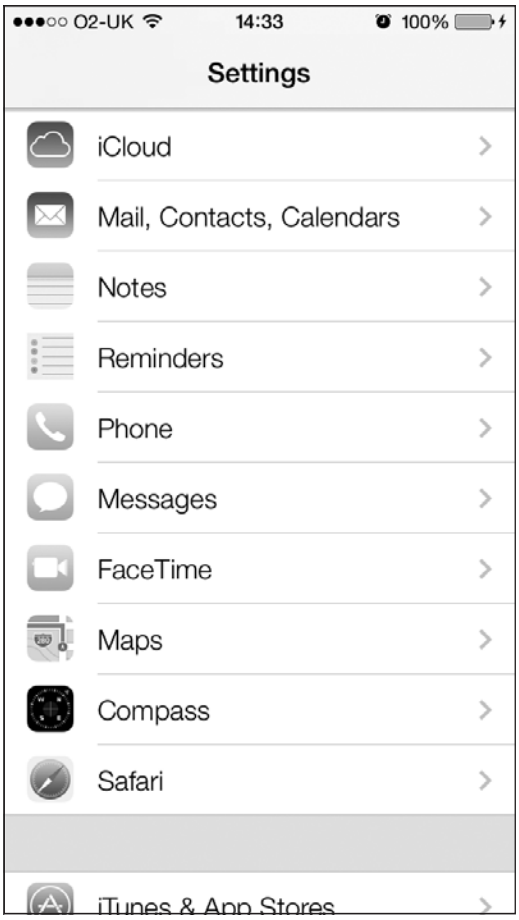


**Рис. 1.45.** Подписи в качестве названий настроек

Как видите, подписи содержат текстовые названия разделов приложения Settings (Настройки), в частности iCloud, Twitter, FaceTime, Safari и т. д.

Чтобы создать подпись, необходимо инстанцировать объект типа `UILabel`. Установка или получение текста для подписи осуществляется с помощью свойства `text`. Итак, определим подпись в файле реализации контроллера нашего вида:

```
#import "ViewController.h"

@interface ViewController ()
@property (nonatomic, strong) UILabel *myLabel;
@end

@implementation ViewController
...
```

А теперь в `viewDidLoad` инстанцируем подпись и сообщаем среде времени исполнения, где следует разместить подпись (эта информация указывается в свойстве

frame) и в какой вид она должна быть добавлена. В данном случае подпись окажется в виде контроллера нашего вида:

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

  CGRect labelFrame = CGRectMake(0.0f,
                                 0.0f,
                                 100.0f,
                                 23.0f);
  self.myLabel = [[UILabel alloc] initWithFrame:labelFrame];
  self.myLabel.text = @"iOS 7 Programming Cookbook";
  self.myLabel.font = [UIFont boldSystemFontOfSize:14.0f];
  self.myLabel.center = self.view.center;
  [self.view addSubview:self.myLabel];

}
```

Теперь запустим приложение и посмотрим, что происходит (рис. 1.46).

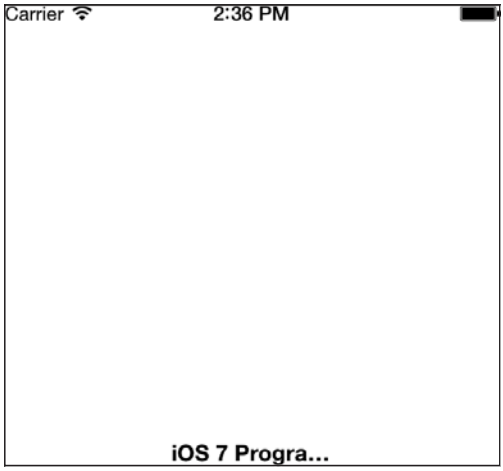


**Рис. 1.46.** Слишком длинная подпись, которая не умещается на экране

Как видите, текст (содержимое) подписи обрезается, а за ним идут точки, поскольку ширины поля для подписи недостаточно для того, чтобы уместился весь текст. Для решения этой проблемы можно было бы увеличить ширину, но что делать с высотой? А что, если мы хотим, чтобы текст переходил на следующую строку. Хорошо, увеличим высоту с 23.0f до 50.0f:

```
CGRect labelFrame = CGRectMake(0.0f,
                               0.0f,
                               100.0f,
                               50.0f);
```

Если сейчас запустить приложение, получится *тот же самый* результат, что и на рис. 1.46. Вы могли бы спросить: «Я увеличил высоту, так почему же текст не

переходит на следующую строку»? Оказывается, у класса `UILabel` есть свойство `numberOfLines`, в котором нужно указать, на сколько строк должен разбиваться текст подписи, если в ширину для нее будет недостаточно места. Если задать здесь значение 3, то вы сообщите программе, что текст подписи должен занимать не более трех строк, если этот текст не умещается в одной строке:

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

  CGRect labelFrame = CGRectMake(0.0f,
                                 0.0f,
                                 100.0f,
                                 70.0f);
  self.myLabel = [[UILabel alloc] initWithFrame:labelFrame];
  self.myLabel.numberOfLines = 3;
  self.myLabel.lineBreakMode = NSLineBreakByWordWrapping;
  self.myLabel.text = @"iOS 7 Programming Cookbook";
  self.myLabel.font = [UIFont boldSystemFontOfSize:14.0f];
  self.myLabel.center = self.view.center;
  [self.view addSubview:self.myLabel];

}
```

Теперь при запуске программы вы получите желаемый результат (рис. 1.47).


**Рис. 1.47.** Подпись, текст которой занимает три строки

Бывает, что вы не знаете, сколько строк понадобится, чтобы отобразить текст подписи. В таких случаях для свойства numberOfLines подписи задается значение 0.

Если вы хотите, чтобы рамка, в которой находится подпись, имела постоянные размеры, а размер шрифта корректировался так, чтобы он входил в отведенные

границы, необходимо задать для свойства adjustsFontSizeToFitWidth подписи значение YES. Например, если высота подписи равна 23.0f, как показано на рис. 1.46, то можно уместить шрифт подписи в этих границах. Вот как это делается:

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

  self.view.backgroundColor = [UIColor whiteColor];
  CGRect labelFrame = CGRectMake(0.0f,
                                 0.0f,
                                 100.0f,
                                 23.0f);
  self.myLabel = [[UILabel alloc] initWithFrame:labelFrame];
  self.myLabel.adjustsFontSizeToFitWidth = YES;
  self.myLabel.text = @"iOS 7 Programming Cookbook";
  self.myLabel.font = [UIFont boldSystemFontOfSize:14.0f];
  self.myLabel.center = self.view.center;
  [self.view addSubview:self.myLabel];

}
```

# 1.18. Оформление UILabel

## Постановка задачи

Требуется возможность оформлять внешний вид подписей — от настройки теней до настройки выравнивания.

## Решение

Пользуйтесь перечисленными далее свойствами класса UILabel в зависимости от стоящей перед вами задачи.

- shadowColor — свойство типа UIColor. Как понятно из названия, оно указывает цвет отбрасываемой тени для подписи. Устанавливая это свойство, вы должны установить и свойство shadowOffset.
- shadowOffset — это свойство типа CGSize. Оно указывает размер отступа между тенью и текстом. Например, если вы зададите для этого свойства значение (1, 0), то тень будет находиться на одну точку правее текста. Если задать значение (1, 2), то тень окажется на одну правее и на одну точку ниже текста. Если же установить значение (-2, -10), то тень будет отображаться на две точки левее и на десять точек выше текста.
- numberOfLines — свойство представляет собой целое число, указывающее, сколько строк текста может включать в себя подпись. По умолчанию значение этого свойства равно 1. Таким образом, любая создаваемая вами подпись по умолчанию может обработать одну строку текста. Если вы хотите сделать подпись из двух строк, задайте для этого свойства значение 2. Если требуется, чтобы в вашем

текстовом поле могло отображаться неограниченное количество текстовых строк, либо вы просто не знаете, сколько строк текста в итоге понадобится отобразить, это свойство должно иметь значение 0. (Лично я нахожу это очень странным. Вместо NSIntegerMax или чего-то подобного в Apple решили обозначать неограниченное количество нулем!)

- lineBreakMode — это свойство относится к типу NSLineBreakMode и указывает способ перехода текста на новую строку внутри текстового поля. Например, если присвоить этому свойству значение NSLineBreakByWordWrapping, то слова разрываться не будут, но если по ширине будет мало места, то текст станет переходить на новую строку. Напротив, если задать для этого свойства значение NSLineBreakByCharWrapping, то при переходе на новую строку может происходить разрыв слова. Вероятно, NSLineBreakByCharWrapping стоит использовать лишь при жестком дефиците места и необходимости уместить на экране как можно больше информации. Я не рекомендую пользоваться этим свойством, если, конечно, вы стремитесь сохранить пользовательский интерфейс аккуратным и четким.
- textAlignment — свойство относится к типу NSTextAlignment и задает выравнивание текста в подписи по горизонтали. Например, для этого свойства можно задать значение NSTextAlignmentCenter, чтобы выровнять текст подписи по центру по горизонтали.
- textColor — это свойство типа UIColor определяет цвет текста подписи.
- font — свойство типа UIFont задает шрифт, которым отображается текст подписи.
- adjustsFontSizeToFitWidth — это свойство типа BOOL. Если оно имеет значение YES, то размер шрифта будет изменяться таким образом, чтобы текст умещался в поле для подписи. Например, когда поле маленькое, а вы хотите записать на нем слишком большой текст. В этом случае среда времени исполнения автоматически уменьшит размер шрифта подписи, чтобы текст гарантированно поместился. Напротив, если для этого свойства задано значение NO, то программа будет действовать в соответствии с актуальной функцией заверстывания строк/слов/символов и текст отобразится не полностью — всего несколько слов.

## Обсуждение

Подписи — одни из простейших компонентов пользовательского интерфейса, которые мы можем использовать в наших приложениях. Но при всей простоте их потенциал очень велик. Поэтому оформление подписей — очень важный фактор, значительно сказывающийся на удобстве использования интерфейса. Поэтому Apple предоставляет нам массу способов оформления экземпляров UILabel. Рассмотрим пример. Мы создаем простое приложение с единственным видом, в котором есть всего один контроллер вида. В центре экрана поместим простую надпись, выполненную огромным шрифтом, — она будет гласить: iOS SDK. Фон вида мы сделаем белым, а цвет тени, отбрасываемой подписью, — светло-серым. Мы убедимся, что тень находится ниже и правее подписи. На рис. 1.48 показан эффект, которого мы стремимся достичь.

А вот и код для этого:

```
#import "ViewController.h"

@interface ViewController ()
@property (nonatomic, strong) UILabel *label;
@end

@implementation ViewController

- (void)viewDidLoad{
  [super viewDidLoad];

  self.label = [[UILabel alloc] init];
  self.label.backgroundColor = [UIColor clearColor];
  self.label.text = @"iOS SDK";
  self.label.font = [UIFont boldSystemFontOfSize:70.0f];
  self.label.textColor = [UIColor blackColor];
  self.label.shadowColor = [UIColor lightGrayColor];
  self.label.shadowOffset = CGSizeMake(2.0f, 2.0f);
  [self.label sizeToFit];
  self.label.center = self.view.center;
  [self.view addSubview:self.label];

}

@end
```

## См. также

Разделы 1.17, 1.26.

# 1.19. Прием пользовательского текстового ввода с помощью UITextField

## Постановка задачи

Необходимо принимать через пользовательский интерфейс программы текст, вводимый пользователем.

## Решение

Воспользуйтесь классом `UITextField`.

## Обсуждение

Текстовое поле очень похоже на подпись тем, что в нем также можно отображать текстовую информацию. Но текстовое поле, в отличие от подписи, может прини-

мать текстовый ввод и во время исполнения. На рис. 1.49 показаны два текстовых поля в разделе Twitter приложения Settings (Настройки) в iPhone.


**Рис. 1.48.** Оформление и отображение подписи на экране

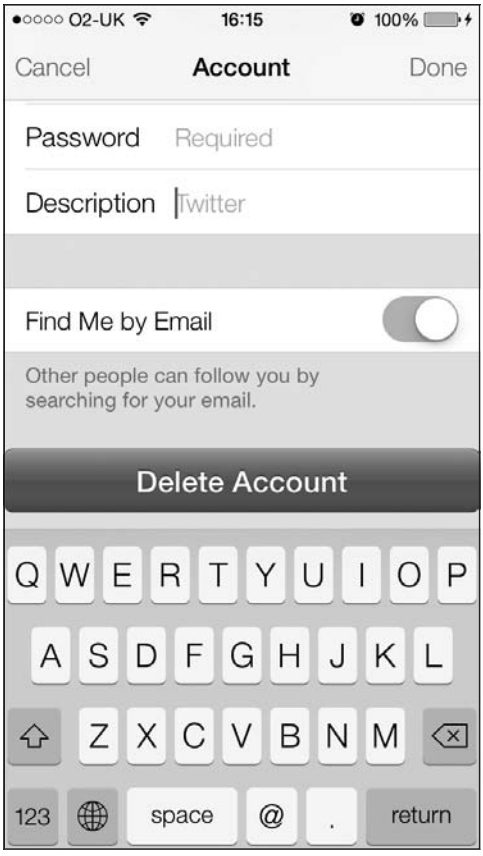


**Рис. 1.49.** Текстовые поля, в которые можно вводить текст

В текстовом поле можно вводить и отображать только одну строку текста. Именно поэтому стандартная высота текстового поля, задаваемая по умолчанию, — всего 31 пункт. Эту высоту нельзя изменить в конструкторе интерфейса, но если вы создаете текстовое поле прямо в коде, то сделать это можно. Тем не менее при изменении высоты не изменяется количество строк, которые можно записать в текстовом поле, — строка всегда всего одна.

Чтобы определить наше текстовое поле, начнем работу с файла реализации контроллера вида:

```
#import "ViewController.h"

@interface ViewController ()
@property (nonatomic, strong) UITextField *myTextField;
```

```
@end

@implementation ViewController

...
```

А потом создадим это текстовое поле:

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

  CGRect textFieldFrame = CGRectMake(0.0f,
                                     0.0f,
                                     200.0f,
                                     31.0f);

  self.myTextField = [[UITextField alloc]
                      initWithFrame:textFieldFrame];

  self.myTextField.borderStyle = UITextBorderStyleRoundedRect;

  self.myTextField.contentVerticalAlignment =
  UIControlContentVerticalAlignmentCenter;

  self.myTextField.textAlignment = NSTextAlignmentCenter;

  self.myTextField.text = @"Sir Richard Branson";
  self.myTextField.center = self.view.center;
  [self.view addSubview:self.myTextField];

}
```

Прежде чем подробно рассматривать код, взглянем на результат его выполнения (рис. 1.50).

При создании этого текстового поля мы использовали различные свойства класса `UITextField`:

- `borderStyle` — свойство имеет тип `UITextBorderStyle` и указывает, как должны отображаться границы текстового поля;
- `contentVerticalAlignment` — это значение типа `UIControlContentVerticalAlignment`, сообщающее текстовому полю, как текст должен отображаться по вертикали в границах этого поля. Если не выровнять текст по центру по вертикали, он по умолчанию отобразится в левом верхнем углу поля;
- `textAlignment` — это свойство имеет тип `UITextAlignment` и указывает выравнивание текста в текстовом поле по горизонтали. В данном примере текст выровнен в текстовом поле по центру и по горизонтали;
- `text` — это свойство доступно как для считывания, так и для записи. То есть можно не только получать из него информацию, но и записывать туда новые данные. Функция считывания возвращает текст, который в данный момент находится в текстовом поле, а функция записи задает для текстового поля то значение, которое вы в ней указываете.

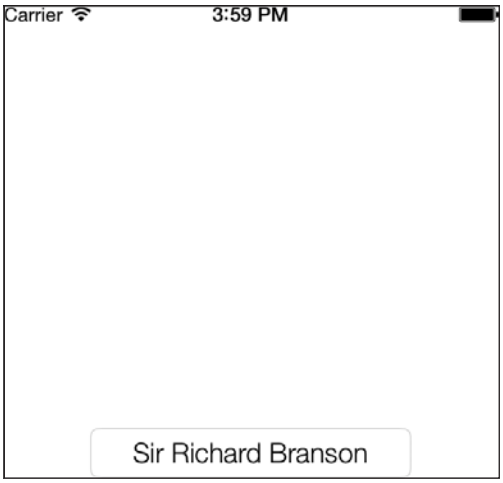


**Рис. 1.50.** Простое текстовое поле, текст в котором выровнен по центру

Текстовое поле посылает сообщения-делегаты своему объекту-делегату. Такие сообщения отправляются, например, когда пользователь начинает изменять (редактировать) информацию в текстовом поле (как-либо изменяет его содержимое) и когда он прекращает взаимодействовать с полем (покидает его). Чтобы получать уведомления об этих событиях, задайте ваш объект в качестве значения свойства `delegate` текстового поля. Делегат текстового поля должен соответствовать протоколу `UITextFieldDelegate`, так что позаботимся об этом:

```
@interface ViewController () <UITextFieldDelegate>
@property (nonatomic, strong) UITextField *myTextField;
@end

@implementation ViewController
```

Нажав и удерживая клавишу Command, щелкните на протоколе `UITextFieldDelegate` в Xcode. Вы увидите методы, которыми позволяет управлять этот протокол. Рассмотрим эти методы, а также укажем, когда они вызываются.

- `textFieldShouldBeginEditing:` — возвращает логическое значение, сообщающее текстовому полю (текстовое поле является параметром этого метода), может ли пользователь редактировать содержащуюся в нем информацию (то есть разрешено это или нет). Возвратите здесь значение `NO`, если не хотите, чтобы пользователь изменял текст в этом поле. Метод запускается, как только пользователь касается этого поля, намереваясь его редактировать (при условии, что в поле допускается редактирование).
- `textFieldDidBeginEditing:` — вызывается, когда пользователь начинает редактировать текстовое поле. Этот метод запускается уже после того, как пользователь коснулся текстового поля, а метод делегата текстового поля `textFieldShouldBeginEditing:` возвратил значение `YES`, сообщив таким образом, что пользователь может редактировать содержимое этого поля.

- `textFieldShouldEndEditing:` — возвращает логическое значение, сообщающее текстовому полю, закончен текущий акт редактирования или нет. Этот метод запускается перед тем, как пользователь собирается покинуть текстовое поле, или после того, как статус активного объекта (First Responder) переходит к другому полю для ввода текста. Если возвратить `NO` от этого метода, то пользователь не сможет перейти в другое текстовое поле и начать вводить текст в него. Виртуальная клавиатура останется на экране.
- `textFieldDidEndEditing:` — вызывается, когда текущий акт редактирования конкретного текстового поля завершается. Это происходит, когда пользователь решает перейти к редактированию какого-то другого текстового поля или нажимает кнопку, предоставленную автором приложения, чтобы убрать с экрана клавиатуру, предназначенную для ввода текста в текстовое поле.
- `textField:shouldChangeCharactersInRange:replacementString:` — вызывается всякий раз, когда текст в текстовом поле изменяется. Возвращаемое значение этого метода — логическое. Если возвращается `YES`, это означает, что текст можно изменить. Если возвращается `NO`, то любые изменения текста в этом поле приняты не будут и даже не произойдут.
- `textFieldShouldClear:` — в каждом текстовом поле есть кнопка *очистки* — обычно это круглая кнопка с крестиком. Когда пользователь нажимает эту кнопку, все содержимое текстового поля автоматически стирается. Если вы предоставляете кнопку для очистки текста, но возвращаете от этого метода значение `NO`, то пользователь может подумать, что ваша программа не работает. Поэтому в данном случае вы должны отдавать себе отчет в том, что делаете. Если пользователь видит кнопку «Стереть», нажимает ее, а текст в поле не исчезает, это очень плохо характеризует программу.
- `textFieldShouldReturn:` — вызывается после того, как пользователь нажимает клавишу Return/Enter, пытаясь убрать клавиатуру с экрана. Текстовое поле должно быть присвоено этому методу в качестве активного элемента.

Объединим этот раздел с разделом 1.17 и создадим динамическую текстовую подпись под нашим текстовым полем. Кроме того, отобразим общее количество символов, введенных в текстовое поле. Начнем с файла реализации:

```
@interface ViewController () <UITextFieldDelegate>
@property (nonatomic, strong) UITextField *myTextField;
@property (nonatomic, strong) UILabel *labelCounter;
@end

@implementation ViewController
```

Теперь создадим текстовое поле с подписью и нужные нам методы делегата текстового поля. Обойдемся без реализации многих методов `UITextFieldDelegate`, так как в этом примере они нам не требуются:

```
- (void) calculateAndDisplayTextFieldLengthWithText:(NSString *)paramText{

  NSString *characterOrCharacters = @"Characters";
  if ([paramText length] == 1){
```

```
      characterOrCharacters = @"Character";
  }

  self.labelCounter.text = [NSString stringWithFormat:@"%lu %@",
                            (unsigned long)[paramText length],
                            characterOrCharacters];
}

- (BOOL) textField:(UITextField *)textField
    shouldChangeCharactersInRange:(NSRange)range
                replacementString:(NSString *)string{

    if ([textField isEqual:self.myTextField]){
        NSString *wholeText =
        [textField.text stringByReplacingCharactersInRange:range
        withString:string];
        [self calculateAndDisplayTextFieldLengthWithText:wholeText];
    }

    return YES;

}

- (BOOL)textFieldShouldReturn:(UITextField *)textField{
   [textField resignFirstResponder];
   return YES;
}

- (void)viewDidLoad{
    [super viewDidLoad];

    CGRect textFieldFrame = CGRectMake(38.0f,
                                       30.0f,
                                       220.0f,
                                       31.0f);

    self.myTextField = [[UITextField alloc]
                        initWithFrame:textFieldFrame];

    self.myTextField.delegate = self;

    self.myTextField.borderStyle = UITextBorderStyleRoundedRect;

    self.myTextField.contentVerticalAlignment =
      UIControlContentVerticalAlignmentCenter;

    self.myTextField.textAlignment = NSTextAlignmentCenter;

    self.myTextField.text = @"Sir Richard Branson";

    [self.view addSubview:self.myTextField];
```

```
    CGRect labelCounterFrame = self.myTextField.frame;
    labelCounterFrame.origin.y += textFieldFrame.size.height + 10;
    self.labelCounter = [[UILabel alloc] initWithFrame:labelCounterFrame];
    [self.view addSubview:self.labelCounter];

    [self calculateAndDisplayTextFieldLengthWithText:self.myTextField.text];

}
```

Мы делаем важное вычисление в методе `textField:shouldChangeCharactersInRange:replacementString:`. Здесь мы объявляем и используем переменную `wholeText`. Когда вызывается этот метод, параметр `replacementString` указывает строку, которую пользователь ввел в текстовое поле. Вы, возможно, полагаете, что пользователь может вводить по одному символу в каждый момент времени, поэтому почему бы не присвоить данному полю значение `char`? Но не забывайте, что пользователь может вставить в текстовое поле целый фрагмент текста, по этой причине данный параметр должен быть строковым. Параметр `shouldChangeCharactersInRange` указывает место в текстовом поле, с которого пользователь начинает вводить текст. Итак, с помощью двух этих параметров мы создаем строку, которая сначала считывает весь текст из текстового поля, а потом использует заданный диапазон, чтобы разместить новый текст рядом со старым. Итак, получается, что вводимый нами текст будет появляться в поле *после* того, как метод `textField:shouldChangeCharactersInRange:replacementString:` возвратит `YES`. На рис. 1.51 показано, как приложение будет выглядеть в эмуляторе.

В текстовом поле может отображаться не только текст, но и *подстановочные (джокерные)* символы. Подстановочный текст отображается *до того*, как пользователь введет в это поле какой-нибудь собственный текст, пока свойство `text` текстового поля является пустым. В качестве подстановочного текста вы можете использовать любую строку, какую хотите, но лучше этим текстом подсказать пользователю, для ввода какой именно информации предназначено данное поле. Многие программисты указывают в подстановочном тексте, значения какого типа может принимать данное поле. Например, на рис. 1.49 в двух текстовых полях (для ввода имени пользователя и пароля) стоит подстановочный текст Required (Обязательно). Можно использовать свойство `placeholder` текстового поля для установки или получения актуального подстановочного текста:

```
CGRect textFieldFrame = CGRectMake(38.0f,
                                   30.0f,
                                   220.0f,
                                   31.0f);
self.myTextField = [[UITextField alloc]
                    initWithFrame:textFieldFrame];

self.myTextField.delegate = self;

self.myTextField.borderStyle = UITextBorderStyleRoundedRect;

self.myTextField.contentVerticalAlignment =
```

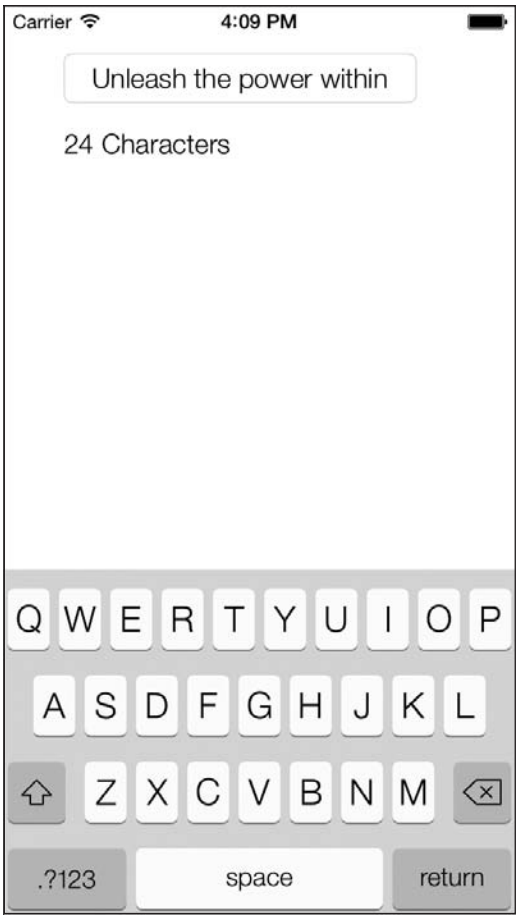


**Рис. 1.51.** Реагирование на сообщения-делегаты текстового поля

```
 UIControlContentVerticalAlignmentCenter;

self.myTextField.textAlignment = UITextAlignmentCenter;

self.myTextField.placeholder = @"Enter text here...";
[self.view addSubview:self.myTextField];
```

Результат показан на рис. 1.52.

У текстовых полей есть два очень приятных свойства, которые называются `leftView` и `rightView`. Они относятся к типу `UIView` и доступны как для чтения, так и для записи. Они проявляются, как понятно из названий, в левой (left) и правой (right) частях текстового поля, когда вы присваиваете им определенный вид. Первое свойство (левый вид) может использоваться, например, при показе курсов валют. В этом случае слева отображается курс валюты страны, в которой проживает пользователь. Поле с этими данными относится к типу `UILabel`. Вот как можно решить такую задачу:


**Рис. 1.52.** Подстановочный текст отображается, когда пользователь еще ничего не ввел в поле

```
UILabel *currencyLabel = [[UILabel alloc] initWithFrame:CGRectZero];
currencyLabel.text = [[[NSNumberFormatter alloc] init] currencySymbol];
currencyLabel.font = self.myTextField.font;
[currencyLabel sizeToFit];
self.myTextField.leftView = currencyLabel;
self.myTextField.leftViewMode = UITextFieldViewModeAlways;
```

Если просто присвоить вид свойству `leftView` или `rightView` текстового поля, то эти виды не появятся автоматически. То, когда они появятся на экране, зависит от режима, управляющего их внешним видом. Данный режим контролируется свойствами `leftViewMode` и `rightViewMode` соответственно. Эти режимы относятся к типу `UITextFieldViewMode`:

```
typedef NS_ENUM(NSInteger, UITextFieldViewMode) {
  UITextFieldViewModeNever,
```

```
  UITextFieldViewModeWhileEditing,
  UITextFieldViewModeUnlessEditing,
  UITextFieldViewModeAlways
}
```

Итак, например, если задать `UITextFieldViewModeWhileEditing` в качестве режима левого вида и присвоить ему значение, то этот вид будет отображаться только в то время, как пользователь редактирует текстовое поле. И наоборот, если задать здесь значение `UITextFieldViewModeUnlessEditing`, левый вид будет отображаться, только пока пользователь *не* редактирует текстовое поле. Как только редактирование начнется, левый вид исчезнет. Теперь запустим наш код в эмуляторе (рис. 1.53).

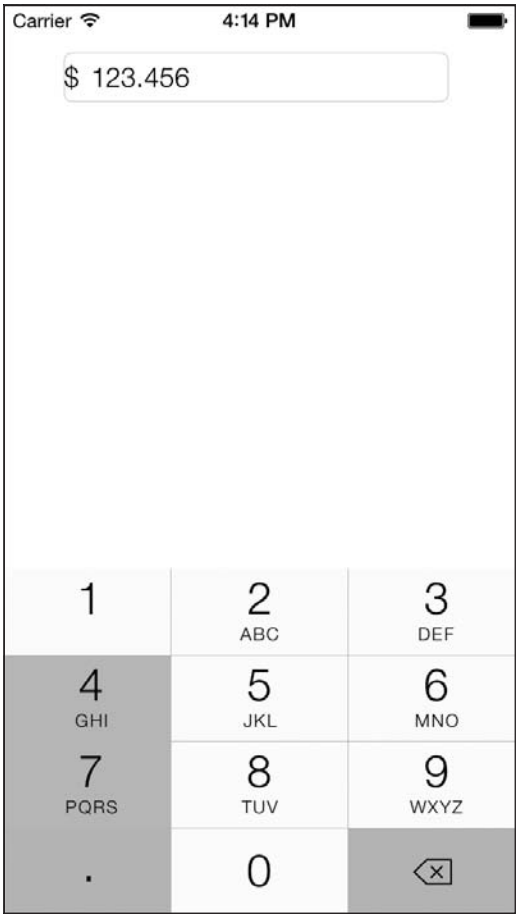


**Рис. 1.53.** Текстовое поле с левым видом

## См. также

Раздел 1.17.

# 1.20. Отображение длинных текстовых строк с помощью UITextView

## Постановка задачи

Требуется отображать в пользовательском интерфейсе несколько строк текста с возможностью прокрутки.

## Решение

Воспользуйтесь классом `UITextView`.

## Обсуждение

Класс `UITextView` позволяет отображать несколько строк текста и создавать прокручиваемое содержимое. Это означает, что если содержимое не умещается в границах текстового вида, то внутренние компоненты этого текстового вида позволяют пользователю прокручивать текст вверх и вниз и просматривать различные его части. В качестве примера текстового вида, входящего в приложение iOS, рассмотрим программу Notes (Блокнот) в iPhone (рис. 1.54).

Создадим текстовый вид и посмотрим, как он работает. Для начала определим текстовый вид в файле реализации контроллера нашего вида:

```
#import "ViewController.h"

@interface ViewController ()
@property (nonatomic, strong) UITextView *myTextView;
@end

implementation ViewController
```

Далее необходимо создать сам текстовый вид. Мы сделаем текстовый вид таким же по размеру, как и вид контроллера вида:

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

  self.myTextView = [[UITextView alloc] initWithFrame:self.view.bounds];
  self.myTextView.text = @"Some text here...";
  self.myTextView.contentInset = UIEdgeInsetsMake(10.0f, 0.0f, 0.0f, 0.0f);
  self.myTextView.font = [UIFont systemFontOfSize:16.0f];
  [self.view addSubview:self.myTextView];

}
```

Запустим приложение в эмуляторе iOS и посмотрим, как оно выглядит (рис. 1.55).


**Рис. 1.54.** Программа Notes (Блокнот) в iPhone, здесь текст отображается в текстовом виде


**Рис. 1.55.** Текстовый вид, занимающий все экранное пространство

Если коснуться текстового поля пальцем, то можно увидеть, как снизу всплывает виртуальная клавиатура. Она довольно крупная и закрывает текстовый вид почти наполовину. То есть если пользователь начнет вводить текст и дойдет примерно до середины окна по вертикали, весь остальной текст, который будет вводиться, *окажется заслоненным клавиатурой* (рис. 1.56).

Чтобы избежать такой ситуации, необходимо слушать определенные уведомления:

- `UIKeyboardWillShowNotification` — система выдает такое уведомление всякий раз, когда клавиатура выводится на экран для работы с каким-либо компонентом: текстовым полем, текстовым видом и т. д.;
- `UIKeyboardDidShowNotification` — система выдает такое уведомление, когда клавиатура отобразится целиком;
- `UIKeyboardWillHideNotification` — система выдает такое уведомление перед тем, как клавиатура скроется из вида;

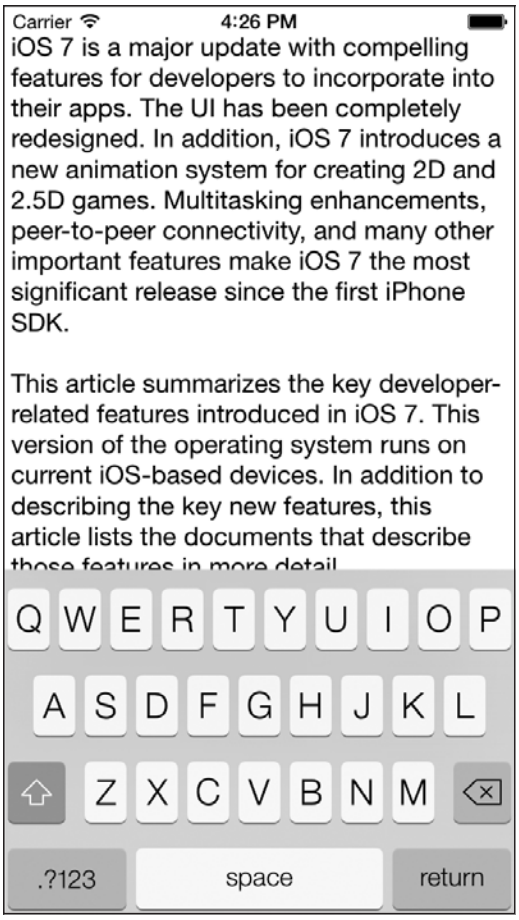


**Рис. 1.56.** Клавиатура, наполовину занимающая текстовый вид

- `UIKeyboardDidHideNotification` — система выдает такое уведомление после того, как клавиатура полностью скроется из вида.

Уведомления клавиатуры содержат словарь, доступный с помощью свойства userInfo. Он указывает границы клавиатуры на экране и относится к типу NSDictionary. В словаре среди прочего имеется ключ UIKeyboardFrameEndUserInfoKey, содержащий объект типа NSValue. В свою очередь, этот объект содержит прямоугольник, ограничивающий размеры клавиатуры, когда она полностью отображена на экране. Эта прямоугольная область обозначается как CGRect.

Наша стратегия такова: нужно узнать, когда клавиатура полностью отобразится, а потом каким-то способом пересчитать размеры нашего текстового вида. Для этого воспользуемся свойством `contentInset` класса `UITextView`, чтобы задать границы контента, содержащегося в текстовом поле, — верхнюю, нижнюю, правую и левую:

```
- (void) handleKeyboardDidShow:(NSNotification *)paramNotification{

  /* Получаем контур клавиатуры. */
  NSValue *keyboardRectAsObject =
    [[paramNotification userInfo]
  objectForKey:UIKeyboardFrameEndUserInfoKey];

  /* Помещаем эту информацию в CGRect. */
  CGRect keyboardRect;

  [keyboardRectAsObject getValue:&keyboardRect];

  /* Задаем нижнюю границу нашего текстового вида так, чтобы он доходил
  ровно до верхней границы клавиатуры. */
  self.myTextView.contentInset =
  UIEdgeInsetsMake(0.0f,
                   0.0f,
                   keyboardRect.size.height,
                   0.0f);
}

- (void) handleKeyboardWillHide:(NSNotification *)paramNotification{
  /* Делаем текстовый вид таким же по размеру, как и вид, содержащий его. */
  self.myTextView.contentInset = UIEdgeInsetsZero;
}

- (void) viewWillAppear:(BOOL)paramAnimated{
  [super viewWillAppear:paramAnimated];

  [[NSNotificationCenter defaultCenter]
   addObserver:self
   selector:@selector(handleKeyboardDidShow:)
   name:UIKeyboardDidShowNotification
   object:nil];

  [[NSNotificationCenter defaultCenter]
   addObserver:self
   selector:@selector(handleKeyboardWillHide:)
   name:UIKeyboardWillHideNotification
   object:nil];

  self.myTextView = [[UITextView alloc] initWithFrame:self.view.bounds];
  self.myTextView.text = @"Some text here...";
  self.myTextView.font = [UIFont systemFontOfSize:16.0f];
  [self.view addSubview:self.myTextView];

}

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

  [[NSNotificationCenter defaultCenter] removeObserver:self];
}
```

В этом коде начинаем наблюдать за клавиатурными уведомлениями в методе `viewWillAppear:` и прекращаем слушать их в методе `viewWillDisappear:`. Важно убрать контроллер вида из списка слушателей, так как вы, вероятно, не хотите получать клавиатурные уведомления, инициируемые контроллером другого вида. Случается, что и при работе в фоновом режиме контроллер вида должен получать уведомления, но это бывает редко. Как правило, нужно прекращать слушание уведомлений в методе `viewWillDisappear:`. Мне не раз доводилось видеть, как программисты портят хорошие приложения, пренебрегая этой простой логикой.

Если вы намереваетесь изменять структуру пользовательского интерфейса, когда клавиатура выводится на экран и когда она с него убирается, то вам никак не обойтись без слушания клавиатурных уведомлений. Сообщения делегата UITextField запускаются всякий раз, когда начинается редактирование текстового поля, независимо от того, есть ли в этот момент на экране клавиатура. Не забывайте, что пользователь может подключить к устройству iOS беспроводную клавиатуру (с помощью Bluetooth). С этой клавиатуры он сможет редактировать содержимое текстовых полей, а также любых других информационных объектов вашего приложения. При подключении клавиатуры по Bluetooth виртуальная клавиатура на экране отображаться не будет. И если в вашем приложении пользовательский интерфейс станет обязательно перестраиваться, как только начинается ввод данных с клавиатуры, то при подключении беспроводной клавиатуры по Bluetooth такая перестройка окажется ненужной.

Теперь, если пользователь попытается ввести какой-либо текст в текстовый вид, клавиатура «выплывет» на экран снизу, и мы присвоим значение высоты клавиатуры в качестве нижней границы содержимого текстового вида. Таким образом, текстовый вид уменьшится в размерах и пользователь сможет вводить в него столько текста, сколько потребуется, — клавиатура не будет заслонять текст.

# 1.21. Добавление кнопок в пользовательский интерфейс с помощью UIButton

## Постановка задачи

Необходимо отобразить в пользовательском интерфейсе кнопку и обрабатывать события касания, связанные с этой кнопкой.

## Решение

Воспользуйтесь классом `UIButton`.

## Обсуждение

Кнопки позволяют пользователям инициировать в приложениях те или иные действия. Например, пакет настроек iCloud в приложении Settings (Настройки) со-

держит кнопку Delete Account (Удалить учетную запись) (рис. 1.57). Если нажать эту кнопку, в приложении iCloud произойдет действие. Оно зависит от конкретного приложения. Не все приложения действуют одинаково, если пользователь нажимает в них кнопку Delete (Удалить). Как мы вскоре увидим, на кнопках могут присутствовать как изображения, так и текст.

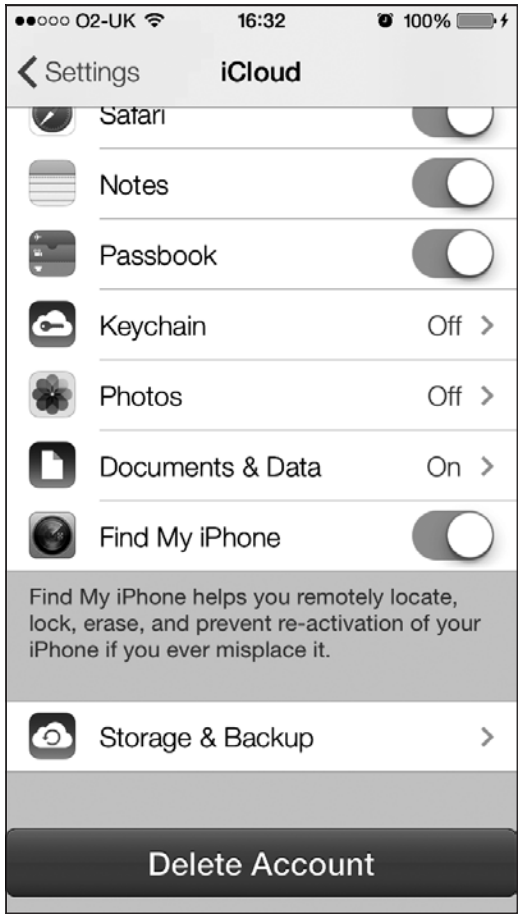


**Рис. 1.57.** Кнопка Delete Account (Удалить учетную запись)

Кнопка может присваивать действия различным инициаторам (триггерам). Например, кнопка может производить одно действие, когда пользователь нажимает ее пальцем, и другое — когда убирает с нее палец. Эти движения становятся действиями, а объекты, реализующие действия, — их целями. Определим кнопку в файле реализации контроллера нашего вида:

```
#import "ViewController.h"

@interface ViewController ()
```

```
@property (nonatomic, strong) UIButton *myButton;
@end

@implementation ViewController
```

По умолчанию высота UIButton в iOS 7 указывается как 44.0f пункта.

Теперь переходим к реализации кнопки (рис. 1.58):

```
- (void) buttonIsPressed:(UIButton *)paramSender{
  NSLog(@"Button is pressed.");
}

- (void) buttonIsTapped:(UIButton *)paramSender{
  NSLog(@"Button is tapped.");
}

- (void)viewDidLoad{
  [super viewDidLoad];

  self.myButton = [UIButton buttonWithType:UIButtonTypeRoundedRect];

  self.myButton.frame = CGRectMake(110.0f,
                                   200.0f,
                                   100.0f,
                                   44.0f);

  [self.myButton setTitle:@"Press Me"
                 forState:UIControlStateNormal];

  [self.myButton setTitle:@"I'm Pressed"
                 forState:UIControlStateHighlighted];

  [self.myButton addTarget:self
                    action:@selector(buttonIsPressed:)
          forControlEvents:UIControlEventTouchDown];

  [self.myButton addTarget:self
                    action:@selector(buttonIsTapped:)
          forControlEvents:UIControlEventTouchUpInside];

  [self.view addSubview:self.myButton];

}
```

В коде из данного примера мы применяем метод `setTitle:forState:` кнопки, задавая для нее два разных заголовка. Заголовок — это надпись на кнопке. В разное время кнопка может находиться в различных состояниях: обычном и утопленном

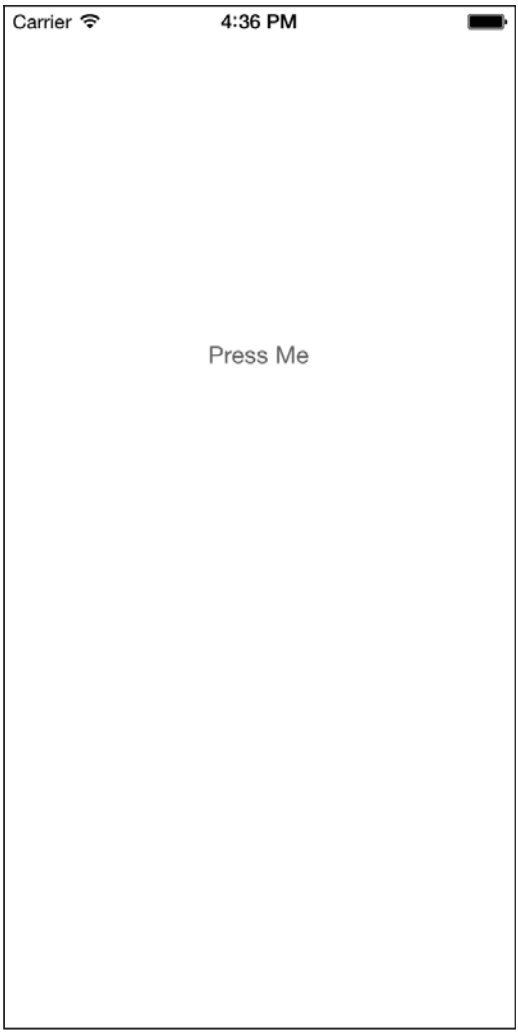


**Рис. 1.58.** В центре экрана находится системная кнопка

(нажатом). В каждом из состояний надпись на ней может меняться. Например, в данном случае, когда пользователь впервые видит кнопку, на ней будет написано Press Me (Нажми меня). А когда он нажмет ее, надпись на кнопке изменится на I'm Pressed (Я нажата).

Аналогичная ситуация складывается и с действиями, инициируемыми кнопкой. Мы используем метод `addTarget:action:forControlEvents:`, чтобы указать для нашей кнопки два действия:

- действие, инициируемое, когда пользователь нажимает кнопку;
- другое действие, происходящее, когда пользователь уже нажал кнопку и убирает палец с экрана. Такое событие называется *окончанием нажатия кнопки* (touch-inside-up).

Еще одна вещь, которую необходимо знать о UIButton, заключается в том, что кнопке обязательно должен быть присвоен тип. Присваивание выполняется путем вызова метода класса buttonWithType на этапе инициализации, как показано в приведенном коде-примере. В качестве параметра этого метода передайте значение типа UIButtonType:

```
typedef NS_ENUM(NSInteger, UIButtonType) {
  UIButtonTypeCustom = 0,
  UIButtonTypeSystem NS_ENUM_AVAILABLE_IOS(7_0),
  UIButtonTypeRoundedRect,
  UIButtonTypeDetailDisclosure,
  UIButtonTypeInfoLight,
  UIButtonTypeInfoDark,
  UIButtonTypeContactAdd,
  UIButtonTypeRoundedRect = UIButtonTypeSystem,
}
```

Кроме того, на кнопке может находиться изображение, которое заменяет ее стандартный внешний вид. Если у вас есть изображение или серия изображений, которые вы хотите присвоить различным состояниям кнопки, убедитесь, что кнопка относится к типу UIButtonTypeCustom. Здесь я подготовил два изображения: одно для обычного состояния кнопки, а другое — для нажатого (утопленного). Сейчас я создам кнопку и присвою ей два этих изображения:

```
UIImage *normalImage = [UIImage imageNamed:@"NormalBlueButton.png"];
UIImage *highlightedImage = [UIImage imageNamed:@"HighlightedBlueButton"];

self.myButton = [UIButton buttonWithType:UIButtonTypeCustom];

self.myButton.frame = CGRectMake(110.0f,
                                 200.0f,
                                 100.0f,
                                 44.0f);

[self.myButton setBackgroundImage:normalImage
                         forState:UIControlStateNormal];
[self.myButton setTitle:@"Normal"
               forState:UIControlStateNormal];

[self.myButton setBackgroundImage:highlightedImage
                         forState:UIControlStateHighlighted];
[self.myButton setTitle:@"Pressed"
               forState:UIControlStateHighlighted];
```

На рис. 1.59 показано, как выглядит приложение, если его запустить в эмуляторе iOS. Чтобы задать фоновое изображение, мы используем относящийся к кнопке метод setBackgroundImage:forState:. Работая с фоновым изображением, мы можем пользоваться методами setTitle:forState: для отображения текста поверх фонового изображения. Если ваше изображение содержит текст и, таким образом, никакой надписи на кнопке не требуется, можете воспользоваться методом setImage:forState: или просто удалить заголовки с кнопки.

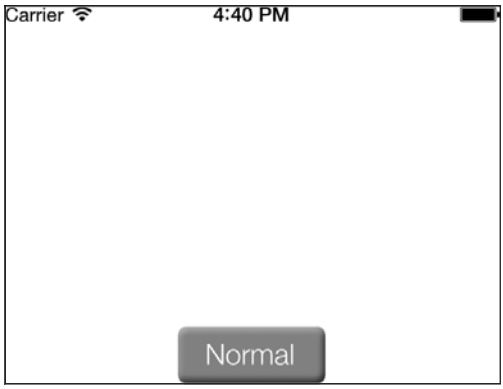


**Рис. 1.59.** Кнопка с фоновым изображением

# 1.22. Показ изображений с помощью UIImageView

## Постановка задачи

Требуется демонстрировать пользователям изображения в графическом интерфейсе программы.

## Решение

Воспользуйтесь классом `UIImageView`.

## Обсуждение

Класс `UIImageView` — один из наименее сложных в iOS SDK. Как вы знаете, существует особый вид, в котором демонстрируются изображения. Чтобы демонстрировать изображения, нужно всего лишь инстанцировать объект типа `UIImageView` и добавлять его к вашим видам. Например, у меня есть картинка Apple MacBook Air и я хочу показать ее в виде для изображений. Начнем с файла реализации контроллера:

```
#import "ViewController.h"

@interface ViewController ()
@property (nonatomic, strong) UIImageView *myImageView;
@end
@implementation ViewController
```

Инстанцируем вид для изображений и разместим в нем изображение:

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

```
    UIImage *macBookAir = [UIImage imageNamed:@"MacBookAir"];
    self.myImageView = [[UIImageView alloc] initWithImage:macBookAir];
    self.myImageView.center = self.view.center;
    [self.view addSubview:self.myImageView];

}
```

Теперь, запустив программу, мы увидим такую картинку, как на рис. 1.60.

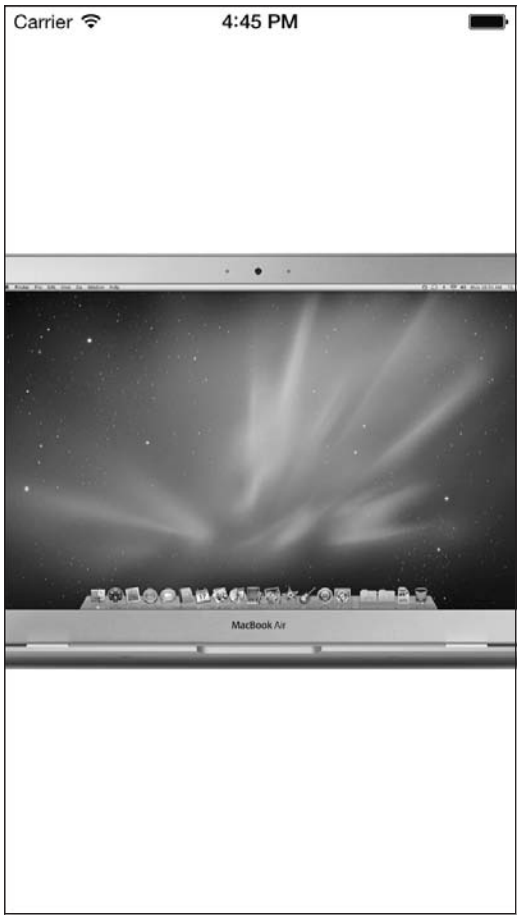


**Рис. 1.60.** Вид с изображением, которое довольно велико и не умещается на экране

Отмечу, что картинка Apple MacBook Air, которую я загружаю в этот вид, имеет разрешение 980 × 519 пикселов и, конечно же, не умещается на экране iPhone. Как решить эту проблему? Для начала нужно убедиться в том, что мы инициализируем наш вид для изображений с помощью метода `initWithFrame:`, а не `initWithImage:`, поскольку второй метод задает высоту и ширину вида с изображением равными высоте и ширине самого изображения. Итак, сначала решим эту проблему:

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

  UIImage *macBookAir = [UIImage imageNamed:@"MacBookAir"];
  self.myImageView = [[UIImageView alloc] initWithFrame:self.view.bounds];
  self.myImageView.image = macBookAir;
  self.myImageView.center = self.view.center;
  [self.view addSubview:self.myImageView];

}
```

Как теперь будет выглядеть наше приложение? Рассмотрим рис. 1.61.


**Рис. 1.61.** Изображение, которое умещается по ширине на экране устройства

Но мы не этого хотели добиться, правда? Действительно, контуры вида с изображением нам теперь подходят, но сама картинка стала отображаться неправильно. Что же можно сделать? Можно решить возникшую проблему, задав для вида с изображением свойство `contentMode`. Это свойство типа `UIContentMode`:

```
typedef NS_ENUM(NSInteger, UIViewContentMode) {
  UIViewContentModeScaleToFill,
  UIViewContentModeScaleAspectFit,
  UIViewContentModeScaleAspectFill,
  UIViewContentModeRedraw,
  UIViewContentModeCenter,
  UIViewContentModeTop,
  UIViewContentModeBottom,
  UIViewContentModeLeft,
  UIViewContentModeRight,
  UIViewContentModeTopLeft,
  UIViewContentModeTopRight,
  UIViewContentModeBottomLeft,
  UIViewContentModeBottomRight,
}
```

Вот описание некоторых наиболее полезных значений из перечня `UIViewContentMode`:

- `UIViewContentModeScaleToFill` — позволяет масштабировать картинку в виде для изображения так, что она целиком заполнит вид в его границах;
- `UIViewContentModeScaleAspectFit` — позволяет гарантировать, что картинка внутри вида с изображением будет иметь правильное соотношение сторон (характеристическое отношение) и будет вписываться в границы вида с изображением;
- `UIViewContentModeScaleAspectFill` — позволяет гарантировать, что картинка внутри вида с изображением будет иметь правильное соотношение сторон и будет вписываться в границы вида с изображением. Чтобы данное значение действовало как следует, необходимо присвоить свойству `clipsToBounds` вида с изображением значение `YES`.

Свойство clipsToBounds вида UIView определяет, должны ли «подокна» этого вида обрезаться, если они выходят за границы содержащего их вида. Можно пользоваться этим свойством, если вы хотите с абсолютной точностью гарантировать, что «подокна» конкретного вида не будут отображаться вне границ содержащего их вида (или что они при необходимости непременно будут выходить за его границы — в зависимости от того, что именно вам требуется).

Итак, чтобы гарантировать, что определенная картинка целиком останется в границах вида с изображением и соотношение сторон этой картинки окажется правильным, нужно применять режим отображения содержимого `UIViewContentModeScaleAspectFit`:

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

  UIImage *macBookAir = [UIImage imageNamed:@"MacBookAir"];
  self.myImageView = [[UIImageView alloc] initWithFrame:self.view.bounds];
  self.myImageView.contentMode = UIViewContentModeScaleAspectFit;
  self.myImageView.image = macBookAir;
```

```
  self.myImageView.center = self.view.center;
  [self.view addSubview:self.myImageView];
}
```

Получается как раз такой результат, которого мы добивались (рис. 1.62).

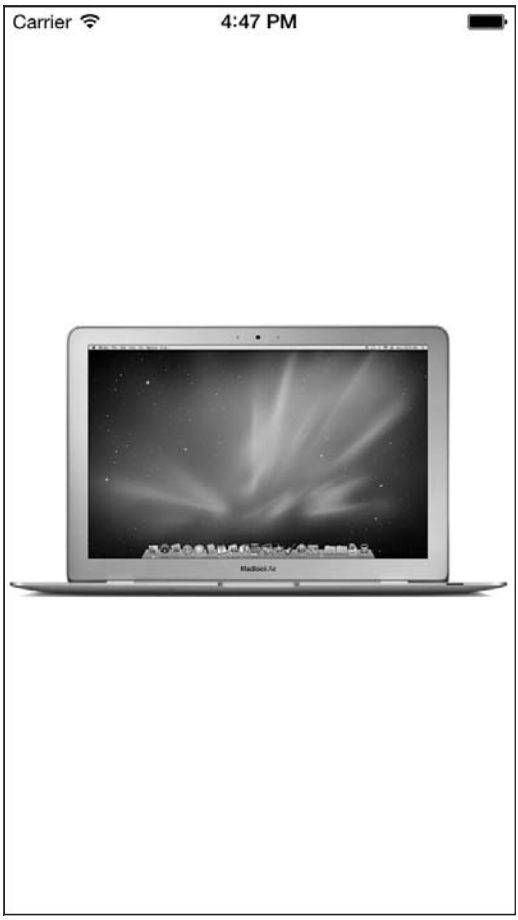


**Рис. 1.62.** Такое соотношение сторон картинки нам подходит

# 1.23. Создание прокручиваемого контента с помощью UIScrollView

## Постановка задачи

Имеется контент, который необходимо отобразить на экране, но вся эта информация занимает больше экранного пространства, чем позволяет одновременно отобразить дисплей нашего устройства.

## Решение

Воспользуйтесь классом `UIScrollView`.

## Обсуждение

Прокручиваемый вид (Scroll View) — одно из очевидных достоинств, которые и делают операционную систему iOS такой удобной. Подобные виды встречаются практически в любых программах. Мы уже познакомились с приложениями Clock (Часы) и Contacts (Контакты). Вы заметили, что их содержимое можно прокручивать вверх и вниз? Да, в этом и заключается магия, присущая видам, о которых пойдет речь в этом разделе.

В сущности, есть всего одна базовая концепция, которую необходимо усвоить в связи с видами, чье содержимое можно прокручивать, — это *размер содержимого*. Учитывая размер содержимого, прокручиваемый вид может адаптироваться к размеру контента, который в нем находится. Размер содержимого — это значение типа `CGSize`, которое указывает высоту и ширину того материала, который наполняет вид с прокручиваемым контентом. Вид с прокручиваемым контентом, как следует из его названия, является подклассом `UIView`. Поэтому вы можете просто добавлять ваши виды к видам с прокручиваемым контентом, пользуясь методом `addSubview:`. Правда, нужно убедиться в том, что размер содержимого для прокручиваемого вида задан правильно. В противном случае эта информация прокручиваться *не будет*.

Найдем для примера большую картинку и загрузим ее в вид с изображением. Я воспользуюсь той самой картинкой, с которой мы работали в разделе 1.22: MacBook Air. Добавлю ее в вид с изображением, который помещу в вид с прокручиваемым контентом. Потом воспользуюсь свойством `contentSize` прокручиваемого вида, чтобы убедиться в том, что размеры этого материала равны размерам изображения (высоте и ширине). Начнем работу с файла реализации контроллера нашего вида:

```
#import "ViewController.h"

@interface ViewController ()
@property (nonatomic, strong) UIScrollView *myScrollView;
@property (nonatomic, strong) UIImageView *myImageView;
@end

@implementation ViewController
```

И поместим вид с изображением внутрь прокручиваемого вида:

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

  UIImage *imageToLoad = [UIImage imageNamed:@"MacBookAir"];
  self.myImageView = [[UIImageView alloc] initWithImage:imageToLoad];
  self.myScrollView = [[UIScrollView alloc] initWithFrame:self.view.bounds];
  [self.myScrollView addSubview:self.myImageView];
```

```
    self.myScrollView.contentSize = self.myImageView.bounds.size;
    [self.view addSubview:self.myScrollView];

}
```

Если теперь загрузить эту программу в эмуляторе iOS, можно убедиться в том, что изображение прокручивается и по горизонтали, и по вертикали. Основная задача в данном случае — найти картинку, которая будет довольно велика и не поместится в пределах экрана. Так, если взять изображение размером 20 × 20 пикселов, то особой пользы от функции прокрутки не будет. Подобную, картинку и не следует помещать в прокручиваемый вид, поскольку в данной ситуации такой вид решительно бесполезен. Прокручивать будет нечего, так как размер экрана больше, чем размер изображения.

`UIScrollView` обладает такой удобной особенностью, как поддержка делегирования. Поэтому такой вид может сообщать приложению о действительно важных событиях с помощью делегата. Делегат для прокручиваемого вида должен отвечать требованиям протокола `UIScrollViewDelegate`. Вот некоторые методы, определяемые в этом протоколе:

- `scrollViewDidScroll:` — вызывается всякий раз, когда содержимое прокручиваемого вида прокручивается;
- `scrollViewWillBeginDecelerating:` — вызывается, когда пользователь прокручивает содержимое вида и отрывает палец от сенсорного экрана в то время, как вид продолжает прокручиваться;
- `scrollViewDidEndDecelerating:` — вызывается, когда прокручивание информации, содержащейся в виде, заканчивается;
- `scrollViewDidEndDragging:willDecelerate:` — вызывается, когда пользователь завершает перетаскивание содержимого в прокручиваемом виде. Этот метод очень напоминает `scrollViewDidEndDecelerating:`, *но* следует помнить, что пользователь может перетаскивать элементы содержимого такого вида и не прокручивая его. Можно просто прикоснуться пальцем к элементу содержимого, переместить палец в другую точку на экране, а потом оторвать палец от экрана, не сдвинув содержимое самого вида ни на миллиметр. Этим перетаскивание и отличается от прокрутки. Прокрутка напоминает перетаскивание, но пользователь «сообщает импульс», приводящий к перемещению содержимого, если снимает палец с экрана, *пока* информация еще прокручивается. То есть пользователь убирает палец, не дождавшись завершения прокрутки. Перетаскивание можно сравнить с тем, как вы удерживаете педаль газа в машине или педаль велосипеда. Продолжая эту аналогию, можно сравнить прокрутку с движением по инерции на машине или велосипеде.

Сделаем предыдущее приложение немного интереснее. Теперь нужно установить уровень яркости картинки в нашем виде с изображением (этот показатель также называется «альфа-уровень» или «альфа-значение») равным `0.50f` (полупрозрачный) на момент, когда пользователь начинает прокрутку изображения, и вернуть этот уровень к значению `1.0f` (непрозрачный) к моменту, когда прокрутка завершается. Сначала обеспечим соответствие протоколу `UIScrollViewDelegate`:

```
#import "ViewController.h"

@interface ViewController () <UIScrollViewDelegate>
@property (nonatomic, strong) UIScrollView *myScrollView;
@property (nonatomic, strong) UIImageView *myImageView;
@end

@implementation ViewController
```

Потом реализуем данную функциональность:

```
- (void)scrollViewDidScroll:(UIScrollView *)scrollView{
  /* Вызывается, когда пользователь совершает прокрутку
  или перетаскивание. */
  self.myScrollView.alpha = 0.50f;
}

- (void)scrollViewDidEndDecelerating:(UIScrollView *)scrollView{
  /* Вызывается только после прокрутки. */
  self.myScrollView.alpha = 1.0f;
}

- (void)scrollViewDidEndDragging:(UIScrollView *)scrollView
  willDecelerate:(BOOL)decelerate{
  /* Гарантируем, что альфа-значение вернется к исходному,
  даже если пользователь просто перетаскивает элементы. */
  self.myScrollView.alpha = 1.0f;
}

- (void)viewDidLoad{
  [super viewDidLoad];

  UIImage *imageToLoad = [UIImage imageNamed:@"MacBookAir"];
  self.myImageView = [[UIImageView alloc] initWithImage:imageToLoad];
  self.myScrollView = [[UIScrollView alloc] initWithFrame:self.view.bounds];
  [self.myScrollView addSubview:self.myImageView];
  self.myScrollView.contentSize = self.myImageView.bounds.size;
  self.myScrollView.delegate = self;
  [self.view addSubview:self.myScrollView];

}
```

Как можно заметить, в прокручиваемых видах имеются *индикаторы*. Индикатор — это тонкая контрольная линия, которая отображается с краю прокручиваемого вида, когда его содержимое прокручивается или перемещается (рис. 1.63).

Индикаторы просто показывают пользователю, как вид расположен в настоящий момент относительно его содержимого (в верхней части, на полпути к низу и т. д.). Внешним видом индикаторов можно управлять, изменяя значение свойства `indicatorStyle`. Например, в следующем коде я делаю индикатор прокручиваемого вида белым:

```
self.myScrollView.indicatorStyle = UIScrollViewIndicatorStyleWhite;
```

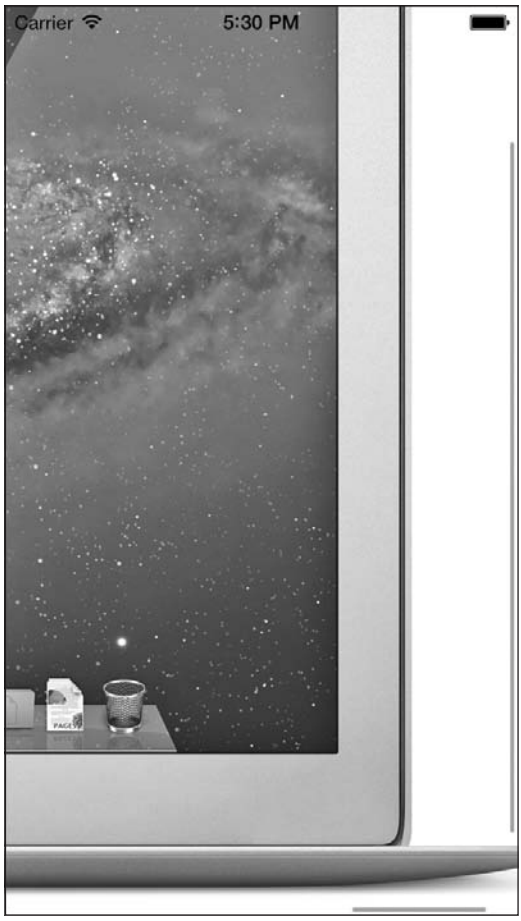


**Рис. 1.63.** Черные индикаторы, появляющиеся справа и снизу прокручиваемого вида

Одна из наиболее замечательных особенностей прокручиваемых видов заключается в том, что в них возможна разбивка на страницы. Она функционально подобна прокрутке, но прокрутка прекращается, как только пользователь переходит на следующую *страницу*. Вероятно, вы уже знакомы с этой функцией, если вам доводилось пользоваться программой Photos (Фотографии) в iPhone или iPad. Просматривая фотографии, можно перемещаться между ними скольжением. Каждое скольжение открывает на экране предыдущую или последующую фотографию. При одном скольжении вы никогда не прокручиваете последовательность до самого начала или до самого конца. Когда начинается прокручивание и вид обнаруживает следующее изображение, прокрутка останавливается на этом изображении и оно начинает подрагивать на экране. Таким образом, анимация прокрутки прерывается. Это и есть разбивка на страницы. Если вы еще не пробовали ее на практике, настоятельно рекомендую попробовать. Весь дальнейший рассказ останется непонятен, если вы не будете представлять, как выглядит приложение, поддерживающее разбивку на страницы.

В следующем примере с кодом я использую три изображения: iPhone, iPad и MacBook Air. Каждое из них я поместил в отдельный вид типа `image view`, а потом добавил эти виды к прокручиваемому виду. Затем включаем разбивку на страницы, задавая для свойства `pagingEnabled` прокручиваемого вида значение `YES`:

```
- (UIImageView *) newImageViewWithImage:(UIImage *)paramImage
                                frame:(CGRect)paramFrame{

    UIImageView *result = [[UIImageView alloc] initWithFrame:paramFrame];
    result.contentMode = UIViewContentModeScaleAspectFit;
    result.image = paramImage;
    return result;

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


    UIImage *iPhone = [UIImage imageNamed:@"iPhone"];
    UIImage *iPad = [UIImage imageNamed:@"iPad"];
    UIImage *macBookAir = [UIImage imageNamed:@"MacBookAir"];

    CGRect scrollViewRect = self.view.bounds;

    self.myScrollView = [[UIScrollView alloc] initWithFrame:scrollViewRect];
    self.myScrollView.pagingEnabled = YES;
    self.myScrollView.contentSize = CGSizeMake(scrollViewRect.size.width *
    3.0f, scrollViewRect.size.height);
    [self.view addSubview:self.myScrollView];

    CGRect imageViewRect = self.view.bounds;
    UIImageView *iPhoneImageView = [self newImageViewWithImage:iPhone
                                                         frame:imageViewRect];
    [self.myScrollView addSubview:iPhoneImageView];

    /* Для перехода на следующую страницу изменяем положение
    следующего вида с изображением по оси X. */
    imageViewRect.origin.x += imageViewRect.size.width;
    UIImageView *iPadImageView = [self newImageViewWithImage:iPad
    frame:imageViewRect];
    [self.myScrollView addSubview:iPadImageView];

    /* Для перехода на следующую страницу изменяем положение
    следующего вида с изображением по оси X. */
    imageViewRect.origin.x += imageViewRect.size.width;
    UIImageView *macBookAirImageView =
    [self newImageViewWithImage:macBookAir
                          frame:imageViewRect];
    [self.myScrollView addSubview:macBookAirImageView];

}
```

Итак, теперь у нас есть три страницы, содержимое которых можно прокручивать (рис. 1.64).


**Рис. 1.64.** Прокрутка содержимого в виде, в котором поддерживается разбивка на страницы

# 1.24. Загрузка веб-страниц с помощью UIWebView

## Постановка задачи

Необходимо динамически загрузить веб-страницу прямо в ваше приложение для iOS.

## Решение

Воспользуйтесь классом `UIWebView`.

## Обсуждение

Веб-вид (Web View) — это окно, которое браузер Safari использует для загрузки в систему iOS информации из Сети. Класс `UIWebView` позволяет использовать в приложениях для iOS всю мощь Safari. Все, что вам нужно сделать, — поместить веб-вид в вашем пользовательском интерфейсе и применить один из методов загрузки:

- `loadData:MIMEType:textEncodingName:baseURL:` — загружает в веб-вид экземпляр класса `NSData`;
- `loadHTMLString:baseURL:` — загружает в веб-вид экземпляр класса `NSString`. Строка должна содержать валидный HTML-код так, чтобы ее мог обработать браузер;
- `loadRequest:` — загружает экземпляр класса `NSURLRequest`. Этот метод пригодится в тех случаях, когда вы хотите загрузить в веб-вид, расположенный в вашем приложении, удаленное содержимое, на которое указывает URL.

Рассмотрим пример. Начнем с файла реализации контроллера нашего вида:

```
#import "ViewController.h"

@interface ViewController ()
@property(nonatomic, strong) UIWebView *myWebView;
@end

@implementation ViewController
```

Теперь я хочу загрузить в веб-вид строку `iOS 7 Programming Cookbook`. Чтобы убедиться в том, что все работает как надо и что наш веб-вид способен отображать насыщенный (форматированный) текст, я на этом не остановлюсь и выделю слово `Cookbook` полужирным шрифтом, а остальной текст оставлю без изменений (рис. 1.65):

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

  self.myWebView = [[UIWebView alloc] initWithFrame:self.view.bounds];
  [self.view addSubview:self.myWebView];

  NSString *htmlString = @"iOS 7 Programming <strong>Cookbook</strong>";
  [self.myWebView loadHTMLString:htmlString
                         baseURL:nil];
}
```

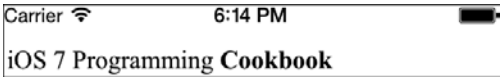


**Рис. 1.65.** Загрузка форматированного текста в веб-вид

Еще один способ работы с веб-видом — загрузка в него удаленного контента, на который указывает URL. Для этого можно пользоваться методом `loadRequest:`.

Перейдем к следующему примеру, в котором загрузим основную страницу сайта Apple в веб-вид, расположенный в нашей программе для iOS (рис. 1.66):

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

  self.myWebView = [[UIWebView alloc] initWithFrame:self.view.bounds];
  self.myWebView.scalesPageToFit = YES;
  [self.view addSubview:self.myWebView];

  NSURL *url = [NSURL URLWithString:@"http://www.apple.com"];
  NSURLRequest *request = [NSURLRequest requestWithURL:url];

  [self.myWebView loadRequest:request];
}
```


**Рис. 1.66.** Веб-вид, в который загружена домашняя страница Apple

Может понадобиться какое-то время, прежде чем в веб-вид загрузится содержимое, которое вы туда передали. Наверное, вы заметили, что при загрузке информации в браузере Safari в левом верхнем углу экрана появляется тонкий индикатор процесса, показывающий, что ваше устройство занято загрузкой контента (рис. 1.67).


**Рис. 1.67.** Индикатор процесса загрузки

В iOS эта задача решается с помощью делегирования. Мы сделаем подписку на делегат веб-вида, и веб-вид будет получать уведомление всякий раз, когда делегат станет загружать контент. Когда загрузка контента завершится, мы получим от веб-вида соответствующее сообщение. Все это мы сделаем, применив свойство `delegate` веб-вида. Делегат веб-вида должен соответствовать протоколу `UIWebViewDelegate`.

Идем дальше. Теперь реализуем в контроллере нашего вида небольшой индикатор процесса. Не забывайте, что индикатор протекающего процесса уже имеется в составе приложения и мы не должны создавать его сами. Управлять этим индикатором можем с помощью метода `setNetworkActivityIndicatorVisible:`, относящегося к `UIApplication`. Итак, начнем с файла реализации контроллера вида:

```
@interface ViewController () <UIWebViewDelegate>
@property(nonatomic, strong) UIWebView *myWebView;
@end

@implementation ViewController
```

Потом перейдем к реализации. Здесь мы будем использовать три метода из тех, которые объявляются в протоколе `UIWebViewDelegate`:

- `webViewDidStartLoad:` — вызывается, как только вид начинает загрузку содержимого;
- `webViewDidFinishLoad:` — вызывается, как только вид заканчивает загрузку содержимого;
- `webView:didFailLoadWithError:` — вызывается, как только вид останавливает загрузку содержимого, например, из-за возникшей ошибки или разрыва сетевого соединения:

```
- (void)webViewDidStartLoad:(UIWebView *)webView{
  [[UIApplication sharedApplication]
  setNetworkActivityIndicatorVisible:YES];
}

- (void)webViewDidFinishLoad:(UIWebView *)webView{
  [[UIApplication sharedApplication] setNetworkActivityIndicatorVisible:NO];
```

```
}

- (void)webView:(UIWebView *)webView didFailLoadWithError:(NSError *)error{
  [[UIApplication sharedApplication] setNetworkActivityIndicatorVisible:NO];
}

- (void)viewDidLoad{
  [super viewDidLoad];

  self.myWebView = [[UIWebView alloc] initWithFrame:self.view.bounds];
  self.myWebView.delegate = self;
  self.myWebView.scalesPageToFit = YES;
  [self.view addSubview:self.myWebView];

  NSURL *url = [NSURL URLWithString:@"http://www.apple.com"];
  NSURLRequest *request = [NSURLRequest requestWithURL:url];

  [self.myWebView loadRequest:request];

}
```

# 1.25. Отображение протекания процессов с помощью UIProgressView

## Постановка задачи

Необходимо отображать на экране индикатор протекания процесса (Progress Bar), отражающий ход выполнения той или иной задачи, например индикатор загрузки файла, скачиваемого с определенного URL.

## Решение

Инстанцируйте вид типа `UIProgressView` и разместите его в другом виде.

## Обсуждение

Вид протекания процесса программисты обычно называют прогресс-баром. Образец такого вида показан на рис. 1.68.

Виды, отображающие протекание процессов, обычно демонстрируются пользователю для показа выполнения задачи с четко определенными начальной и конечной точками. Примером такой задачи является, например, скачивание 30 файлов. Очевидно, что такая задача будет выполнена, когда все 30 файлов будут скопированы на устройство. Вид, отображающий протекание процесса, является экземпляром `UIProgressView` и инициализируется с помощью специального метода-инициализатора данного класса — `initWithProgressViewStyle:`. В качестве параметра данный метод принимает стиль (оформление) панели протекания, которую предполагается создать.

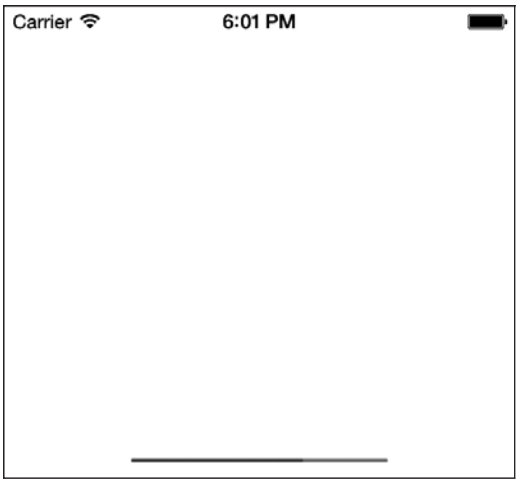


**Рис. 1.68.** Простой вид с индикатором протекания процесса

Этот параметр относится к типу `UIProgressViewStyle` и, соответственно, может иметь одно из следующих значений:

- `UIProgressViewStyleDefault` — это стандартное оформление вида протекания процесса. Именно в этом стиле оформлен вид, показанный на рис. 1.68;
- `UIProgressViewStyleBar` — напоминает `UIProgressViewStyleDefault`, но предназначено для использования с видами отображения протекания процессов, добавляемыми на панель инструментов.

Экземпляр `UIProgressView` определяет свойство под названием `progress` (типа `float`). Это свойство сообщает системе iOS, как должна отображаться полоса в виде, отражающем протекание процесса. Значение этого свойства должно быть в диапазоне от `0` до `1.0`. Если сообщается значение `0`, то заполнение индикатора состояния еще не началось. Значение `1.0` соответствует 100%-ной завершенности. Степень прогресса, показанная на рис. 1.68, составляет `0.5` (или 50 %).

Чтобы научиться создавать виды, отражающие протекание процессов, создадим вид, похожий на тот, что приведен на рис. 2.74. Начинаем с главного — определяем свойство для вида протекания процесса:

```
#import "ViewController.h"

@interface ViewController ()
@property (nonatomic, strong) UIProgressView *progressView;
@end

@implementation ViewController
```

Далее инстанцируем объект типа `UIProgressView`:

```
- (void)viewDidLoad{

  [super viewDidLoad];
```

```
    self.progressView = [[UIProgressView alloc]
    initWithProgressViewStyle:UIProgressViewStyleBar];
    self.progressView.center = self.view.center;
    self.progressView.progress = 20.0f / 30.0f;

    [self.view addSubview:self.progressView];

}
```

Итак, создать вид протекания процесса совсем не сложно. В сущности, нужно просто правильно отобразить ход процесса, так как свойство `progress` данного вида должно иметь значение в диапазоне от `0` до `1.0`, то есть нормализованное значение. Итак, если вам предстоит решить 30 задач и вы уже выполнили 20 из них, то нужно присвоить свойству `progress` вида протекания процесса результат следующего равенства:

```
self.progressView.progress = 20.0f / 30.0f;
```

Значения 20 и 30 передаются данному равенству как значения с плавающей точкой, поскольку компилятору нужно сообщить, что операция деления будет производиться над числами с плавающей точкой и в результате деления получится десятичная дробь. Если приказать компилятору поместить в свойстве progress вида протекания процесса целочисленное деление 20/30, то вы получите целочисленный результат 0. Это происходит потому, что компилятор выполняет целочисленное деление, отсекая полученный результат до ближайшего предшествующего целого числа. Короче говоря, на индикаторе протекания действия прогресс все время будет оставаться нулевым, пока процесс не завершится и частное от деления 30/30 не станет равно 1. Пользователю такой индикатор загрузки будет ни к чему.

# 1.26. Создание и отображение текстов с оформлением

## Постановка задачи

Требуется возможность отображать в элементах вашего пользовательского интерфейса насыщенный форматированный текст, избегая при этом необходимости создавать отдельный компонент пользовательского интерфейса для каждого атрибута. Например, может потребоваться отобразить в `UILabel` предложение, в котором всего одно слово записано полужирным шрифтом.

## Решение

Создайте экземпляр класса `NSAttributedString` или его изменяемого варианта, `NSMutableAttributedString`, и либо задайте его как текст компонента пользовательского интерфейса (например, как текст подписи `UILabel`) с помощью специального строкового свойства, снабженного атрибутами, либо просто воспользуйтесь встроенными методами атрибутированной строки для отрисовки текста на холсте.

## Обсуждение

О насыщенном тексте слагают легенды. Многим из наших коллег-программистов приходилось сталкиваться с необходимостью отображения в пользовательском интерфейсе такой текстовой строки, в которой применяется сразу несколько видов форматирования. Например, в одной строке может понадобиться одновременно вывести и обычный текст, и курсив, причем курсивом будет записано всего одно слово. Возможно, одно из слов в предложении потребуется подчеркнуть. Для этого некоторые пытаются использовать веб-виды (Web Views), но это решение не является оптимальным, поскольку веб-виды довольно медленно отображают свой контент и неизбежно негативно воздействуют на производительность приложения. В iOS 7 можно приступать к применению атрибутированных строк. Не знаю, почему Apple решила внедрить такую возможность в iOS только сейчас, ведь Mac-разработчики пользуются атрибутированными строками уже довольно давно.

Прежде чем приступить к основной части раздела, я хотел бы четко пояснить, что понимается под термином «атрибутированная строка». Взгляните на рис. 1.69. Мы собираемся написать программу, которая будет достигать именно такого эффекта.


**Рис. 1.69.** Атрибутированная строка отображена на экране в простой подписи

Необходимо отметить, что этот текст отображается в одном экземпляре класса UILabel.

Итак, что мы видим в этом примере? Перечислю.

- Текст `iOS` имеет следующие атрибуты:
  - полужирный шрифт размером 60 точек;
  - черный цвет фона;
  - красный цвет шрифта.

- Текст SDK имеет следующие атрибуты:
  - полужирный шрифт размером 60 точек;
  - белый цвет шрифта;
  - светло-серую тень;
  - красный цвет фона.

Удобнее всего создавать атрибутированные строки с помощью метода initWithString:, относящегося к изменяемому классу NSMutableAttributedString, и передавать этому методу экземпляр NSString. Так создается атрибутированная строка без каких-либо атрибутов. Затем, чтобы присвоить атрибуты различным частям строки, мы воспользуемся методом setAttributes:range: класса NSMutableAttributedString. Этот метод принимает два параметра:

- setAttributes — словарь, ключи которого являются символьными атрибутами и значение каждого ключа зависит от самого ключа. Вот наиболее важные ключи, которые можно задать в этом словаре:
  - NSFontAttributeName — значение этого ключа является экземпляром UIFont и определяет шрифт для того или иного фрагмента строки;
  - NSForegroundColorAttributeName — значение этого ключа относится к типу UIColor и определяет цвет шрифта определенного фрагмента строки;
  - NSBackgroundColorAttributeName — значение этого ключа относится к типу UIColor и определяет цвет фона, на котором будет отрисовываться определенный фрагмент строки;
  - NSShadowAttributeName — значение этого ключа должно быть экземпляром NSShadow и задавать тень, которую будет отбрасывать определенный фрагмент строки;
- range — значение типа NSRange, определяющее начальную точку и длину группы символов, к которой вы хотите применить указанные атрибуты.

Чтобы просмотреть все ключи, которые можно передавать этому методу, просто изучите онлайновую документацию Apple по классу NSMutableAttributedString. Я не буду помещать здесь ссылку на документацию, так как Apple может рано или поздно изменить эту ссылку, а вот поиск вас точно не подведет.

Разобьем наш пример на два словаря с атрибутами. Словарь атрибутов для слова iOS создается в коде таким образом:

```
NSDictionary *attributesForFirstWord = @{
  NSFontAttributeName : [UIFont boldSystemFontOfSize:60.0f],
  NSForegroundColorAttributeName : [UIColor redColor],
  NSBackgroundColorAttributeName : [UIColor blackColor]
};
```

А слово SDK создается с помощью следующих атрибутов:

```
NSShadow *shadow = [[NSShadow alloc] init];
shadow.shadowColor = [UIColor darkGrayColor];
```

```
shadow.shadowOffset = CGSizeMake(4.0f, 4.0f);

NSDictionary *attributesForSecondWord = @{
  NSFontAttributeName : [UIFont boldSystemFontOfSize:60.0f],
  NSForegroundColorAttributeName : [UIColor whiteColor],
  NSBackgroundColorAttributeName : [UIColor redColor],
  NSShadowAttributeName : shadow
};
```

Собрав все вместе, получаем следующий код, который не только создает нашу подпись, но и задает для нее атрибутированный текст:

```
#import "ViewController.h"

@interface ViewController ()
@property (nonatomic, strong) UILabel *label;
@end

@implementation ViewController

- (NSAttributedString *) attributedText{

  NSString *string = @"iOS SDK";

  NSMutableAttributedString *result = [[NSMutableAttributedString alloc]
                                       initWithString:string];
  NSDictionary *attributesForFirstWord = @{
    NSFontAttributeName : [UIFont boldSystemFontOfSize:60.0f],
    NSForegroundColorAttributeName : [UIColor redColor],
    NSBackgroundColorAttributeName : [UIColor blackColor]
  };

  NSShadow *shadow = [[NSShadow alloc] init];
  shadow.shadowColor = [UIColor darkGrayColor];
  shadow.shadowOffset = CGSizeMake(4.0f, 4.0f);

  NSDictionary *attributesForSecondWord = @{
    NSFontAttributeName : [UIFont boldSystemFontOfSize:60.0f],
    NSForegroundColorAttributeName : [UIColor whiteColor],
    NSBackgroundColorAttributeName : [UIColor redColor],
    NSShadowAttributeName : shadow
  };

  /* Находим фрагмент iOS в целой строке и задаем атрибуты для этого фрагмента */
  [result setAttributes:attributesForFirstWord
                  range:[string rangeOfString:@"iOS"]];

  /* Делаем то же самое со строкой SDK */
  [result setAttributes:attributesForSecondWord
                  range:[string rangeOfString:@"SDK"]];

  return [[NSAttributedString alloc] initWithAttributedString:result];
```

```
}

- (void)viewDidLoad{
  [super viewDidLoad];

  self.label = [[UILabel alloc] init];
  self.label.backgroundColor = [UIColor clearColor];
  self.label.attributedText = [self attributedText];
  [self.label sizeToFit];
  self.label.center = self.view.center;
  [self.view addSubview:self.label];

}

@end
```

## См. также

Разделы 1.17 и 1.18.

# 1.27. Представление видов «Основной — детали» с помощью UISplitViewController

## Постановка задачи

Необходимо максимально эффективно использовать большой экран iPad, представив на нем два расположенных рядом контроллера видов.

## Решение

Воспользуйтесь классом `UISplitViewController`.

## Обсуждение

Контроллеры видов split view (будем называть эти виды разделенными экранами) есть только в iPad. Если вы работаете с iPad, то, вероятно, уже сталкивались с ними. Можно просто открыть приложение Settings (Настройки) в альбомном режиме и посмотреть. Видите, какой контроллер разделенного экрана показан на рис. 1.70?

У контроллера разделенного экрана есть левая и правая стороны. Слева отображаются основные настройки. При нажатии каждой из этих настроек открываются детали этого элемента, которые мы видим в правой части разделенного экрана.

Даже не пытайтесь инстанцировать объект типа UISplitViewController на каком-нибудь устройстве, кроме iPad. В результате вы получите исключение.

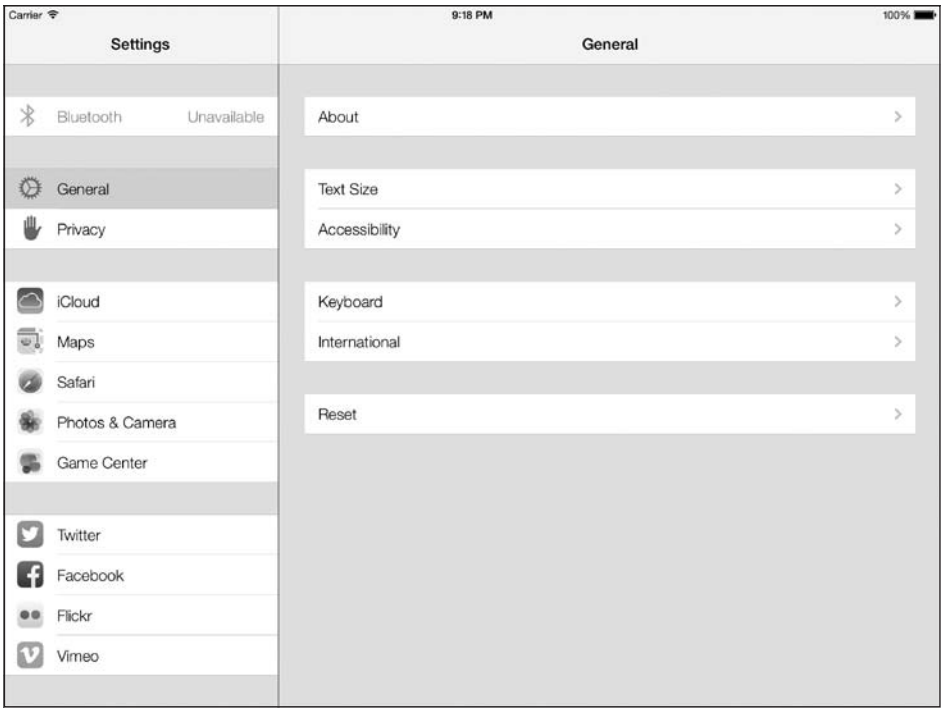


**Рис. 1.70.** Контроллер с разделенным экраном в приложении Settings (Настройки) в iPad

Apple предельно упростила процесс создания приложений, в основе которых лежит работа с разделенными экранами. Чтобы создать собственное приложение такого рода, просто выполните следующие шаги.

1. В Xcode перейдите в меню File (Файл) и выполните New ▸ New Project (Новый ▸ Новый проект).
2. В окне New Project (Новый проект) выберите слева iOS ▸ Application (iOS ▸ Приложение), а потом укажите вариант Master-Detail Application (Приложение «Основной — детали») (рис. 1.71) и нажмите Next (Далее).
3. На следующем экране выберите название вашего продукта и убедитесь в том, что для семейства устройств указан параметр Universal (Универсальное). Мы хотим, чтобы создаваемое приложение могло работать и на iPhone, и на iPad (рис. 1.72). Сделав это, нажмите Next (Далее).
4. Теперь выберем место для сохранения проекта. Сделав это, нажмите кнопку Create (Создать).

Итак, проект создан. На кнопке поэтапного выбора Scheme (Схема), расположенной в левом верхнем углу, должно быть указано, что приложение будет работать в эмуляторе iPad, а не в эмуляторе iPhone. Если в Xcode создается универсальное приложение «Основной — детали», то Xcode обеспечивает возможность работы с этим приложением и на iPhone, но при запуске приложения на iPhone

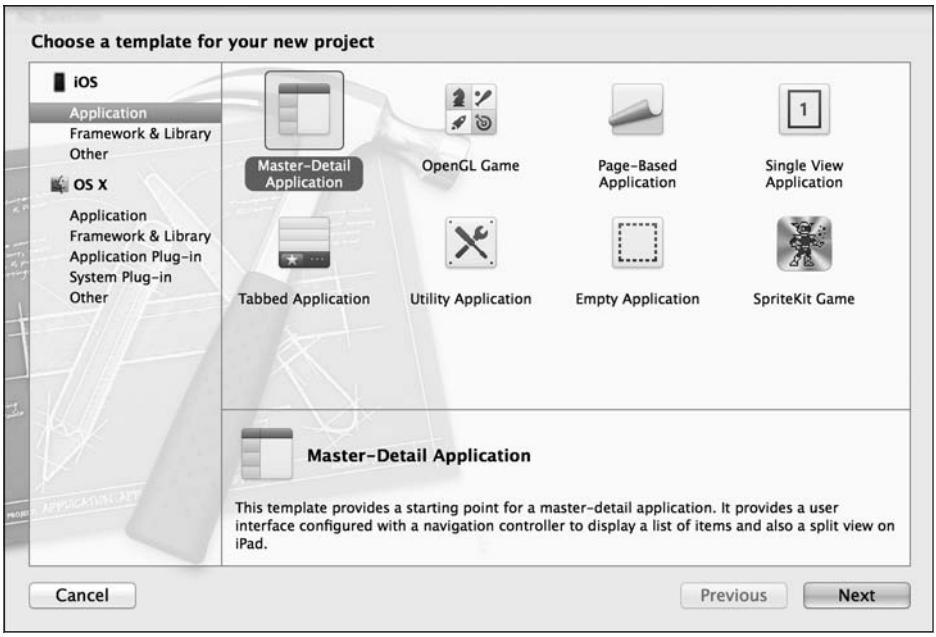


**Рис. 1.71.** Выбираем в Xcode шаблон приложения «Основной — детали»

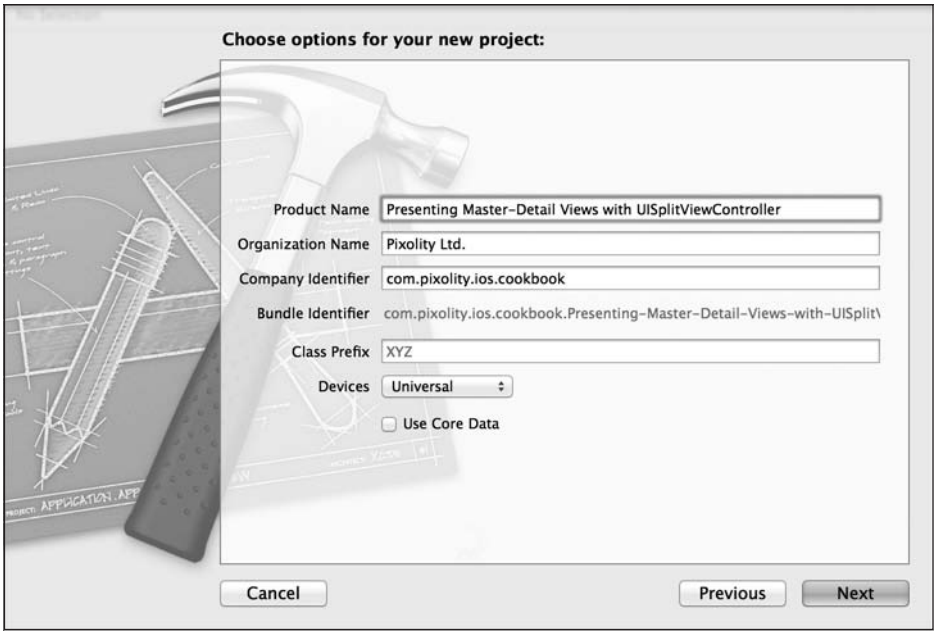


**Рис. 1.72.** Задаем в Xcode настройки проекта «Основной — детали»

структура его будет иной, нежели при запуске на iPad. В приложении окажется навигационный контроллер, внутри которого будет контроллер вида. Если то же самое приложение запустить на iPad, то мы увидим разделенный экран, в котором будут расположены два контроллера вида.

В шаблоне проекта с разделенным экраном есть два файла, о которых следует поговорить отдельно:

- `MasterViewController` — контроллер основного вида, располагающегося в левой части разделенного экрана в iPad. В iPhone это первый контроллер, который увидит пользователь;
- `DetailViewController` — контроллер вида с деталями, который отображается в правой части разделенного экрана на iPad. В iPhone это тот контроллер, который занимает верхнюю позицию в стеке, как только пользователь выбирает любой элемент в корневом (первом, основном) контроллере вида.

Теперь нужно подумать, как будет выглядеть обмен информацией между экраном основных параметров и экраном деталей. Хотите ли вы организовать такой обмен информацией через делегат приложения или желаете, чтобы основной вид посылал сообщения непосредственно виду с деталями? Это зависит от вас.

Если запустить такое приложение в эмуляторе iPad, то в альбомном режиме мы увидим контроллеры основного вида и вида с деталями в разделенном экране, но если изменить ориентацию на книжную, то вид с основными параметрами исчезнет и на его месте появится навигационная кнопка Master (Основной). Она будет располагаться в левой верхней части навигационной панели контроллера с детальной информацией. Хотя это и неплохой вариант, но мы ожидали иного, так как сравниваем наш проект с приложением Settings (Настройки) из iPad. Если в iPad повернуть экран с приложением Settings (Настройки) так, чтобы он приобрел книжную ориентацию, то на экране все равно останутся оба контроллера видов: и с основной информацией, и с деталями. Как нам добиться такого результата? Оказывается, Apple предлагает API (интерфейс программирования приложений), с помощью которого как раз и можно решить такую задачу. Просто переходим в файл `DetailViewController.m` и реализуем следующий метод:

```
- (BOOL) splitViewController:(UISplitViewController *)svc
    shouldHideViewController:(UIViewController *)vc
               inOrientation:(UIInterfaceOrientation)orientation{
  return NO;
}
```

Если вернуть из этого метода значение `NO`, iOS *не будет скрывать* контроллер основного вида при любой ориентации и оба контроллера — как с основными опциями, так и с их деталями — будут отображаться и в альбомной, и в книжной ориентации. Теперь, реализовав упомянутый метод, мы сможем обойтись без двух следующих методов:

```
- (void)splitViewController:(UISplitViewController *)splitController
     willHideViewController:(UIViewController *)viewController
          withBarButtonItem:(UIBarButtonItem *)barButtonItem
       forPopoverController:(UIPopoverController *)popoverController{
barButtonItem.title = NSLocalizedString(@"Master", @"Master");
```

```
[self.navigationItem setLeftBarButtonItem:barButtonItem animated:YES];
self.masterPopoverController = popoverController;
}

- (void)splitViewController:(UISplitViewController *)splitController
     willShowViewController:(UIViewController *)viewController
  invalidatingBarButtonItem:(UIBarButtonItem *)barButtonItem{
[self.navigationItem setLeftBarButtonItem:nil animated:YES];
self.masterPopoverController = nil;
}
```

Эти методы требовались нам просто для управления кнопкой из навигационной панели, но теперь мы больше не пользуемся ею и можем избавиться от этих методов. Их можно просто закомментировать или вообще удалить из файла `DetailViewController.m`.

Заглянув на заголовочный файл контроллера вашего основного вида, вы увидите там нечто подобное:

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

@class DetailViewController;

@interface MasterViewController : UITableViewController

@property (strong, nonatomic) DetailViewController *detailViewController;

@end
```

Как видите, в контроллере основного вида стоит ссылка на контроллер вида с деталями. С помощью этой связи мы можем сообщать контроллеру вида с деталями о сделанном выборе, а также передавать ему другие значения — об этом чуть позже.

По умолчанию если вы запустите приложение в эмуляторе iPad, то увидите пользовательский интерфейс, очень напоминающий тот, что показан на рис. 1.73. В стандартной реализации, которую Apple предоставляет нам с контроллером основного вида, содержится изменяемый массив. Этот массив заполняется экземплярами `NSDate` всякий раз, когда вы нажимаете кнопку «плюс» (+) на навигационной панели в этом контроллере вида. Стандартная реализация очень проста, и вы можете ее модифицировать, немного разобравшись в табличных видах. О том, что такое табличные виды и как они заполняются, подробно рассказано в главе 4.

# 1.28. Организация разбивки на страницы с помощью UIPageViewController

## Постановка задачи

Необходимо создать приложение, работающее по принципу iBooks, где пользователь может листать страницы, как в настоящей книге. Таким образом мы собираемся обеспечить пользователю интуитивно понятную и реалистичную работу с программой.

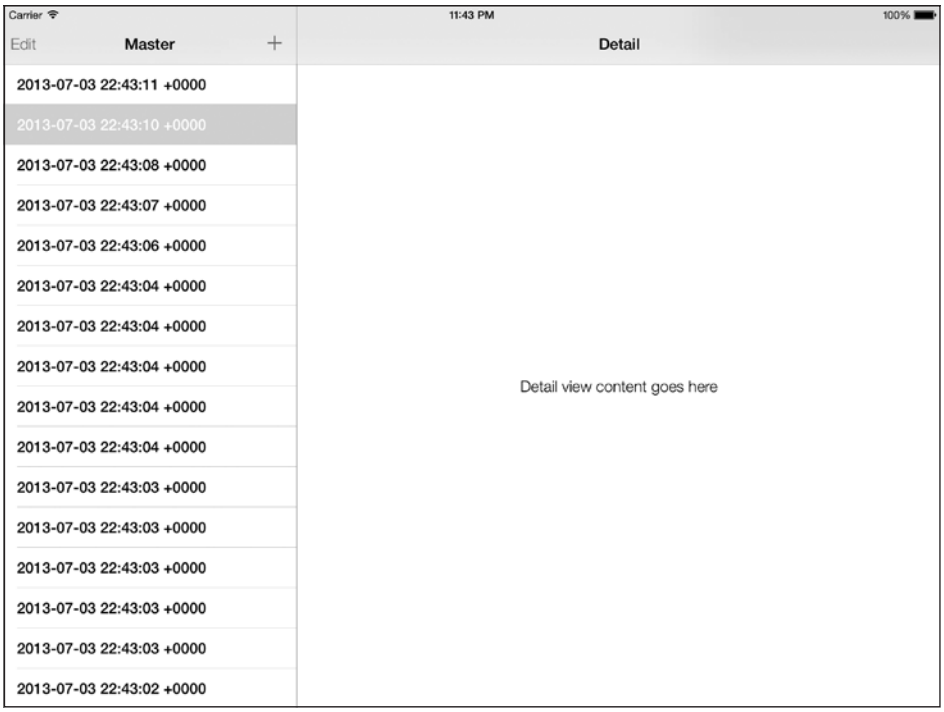


**Рис. 1.73.** Контроллер пустого вида с разделенным экраном, работающий в эмуляторе iPad

## Решение

Воспользуйтесь `UIPageViewController`.

## Обсуждение

В среде разработки Xcode есть шаблон для создания контроллеров с постраничной организацией. Перед тем как изучать этот раздел и узнать, что же они собой представляют, стоит просто посмотреть, как они выглядят. Итак, выполните следующие шаги, чтобы в вашем приложении можно было использовать контроллеры видов с постраничной организацией.

Контроллеры видов с постраничной организацией работают как в iPhone, так и в iPad.

1. В Xcode перейдите в меню File (Файл) и выберите New ▸ New Project (Новый ▸ Новый проект).
2. Убедитесь, что в левой части окна New Project (Новый проект) выбрана операционная система iOS, а далее — команда Application (Приложение). Сделав это, укажите справа шаблон Page-Based Application (Приложение с постраничной организацией) (рис. 1.74) и нажмите Next (Далее).


**Рис. 1.74.** Создание в Xcode приложения с постраничной организацией

3. Теперь выберите имя продукта и убедитесь в том, что указанное вами семейство устройств (Device) является универсальным (Universal). Это необходимо сделать, поскольку, как правило, ваше приложение потребуется использовать и на iPhone, и на iPad (рис. 1.75). Сделав это, нажмите Next (Далее).

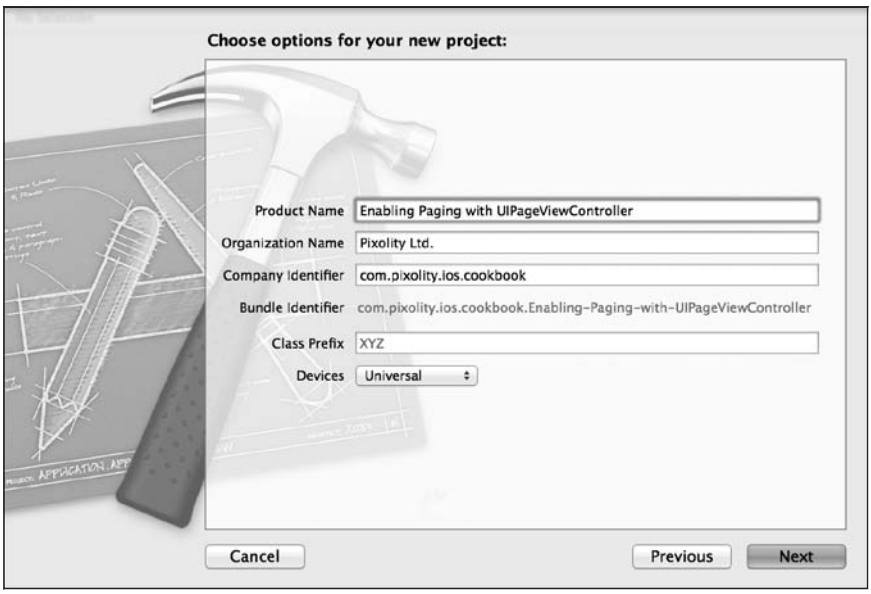


**Рис. 1.75.** Задаем настройки проекта для приложения с постраничной организацией

4. Выберите, где вы хотите сохранить проект. Сделав это, нажмите кнопку Create (Создать). Итак, вы успешно создали проект.

Теперь можете убедиться в том, что Xcode уже создала для вашего проекта несколько классов. Кратко рассмотрим каждый из них:

- класс делегата — делегат приложения просто создает экземпляр класса `RootViewController` и представляет его пользователю. Для iPad используется один архив XIB, для iPhone — другой, но оба они при работе опираются на вышеупомянутый класс;
- `RootViewController` — создает экземпляр `UIPageViewController` и добавляет к себе этот контроллер вида. Поэтому пользовательский интерфейс контроллера данного вида — это фактически смесь двух контроллеров видов, самого `RootViewController` и `UIPageViewController`;
- `DataViewController` — для каждой страницы в контроллере постраничного вида пользователю предлагается по одному экземпляру данного класса. Данный класс является подклассом `UIViewController`;
- `ModelController` — это обычный подкласс `NSObject`, соответствующий протоколу `UIPageViewControllerDataSource`. Этот класс является источником данных для контроллера вида-страницы.

Итак, мы видим, что у контроллера страничного вида есть и делегат, и источник данных. При использовании стандартного шаблона для приложений с постраничной организацией, входящего в состав Xcode, корневой контроллер вида становится делегатом, а контроллер модели — источником данных для контроллера страничного вида. Чтобы понять, как же на самом деле работает контроллер вида-страницы, необходимо разобраться в протоколах, регламентирующих в нем процессы делегирования и обращения к источнику данных. Начнем с протокола делегата, `UIPageViewControllerDelegate`. В этом протоколе есть два важных метода:

```
- (void)pageViewController:(UIPageViewController *)pageViewController
       didFinishAnimating:(BOOL)finished
  previousViewControllers:(NSArray *)previousViewControllers
      transitionCompleted:(BOOL)completed;

- (UIPageViewControllerSpineLocation)pageViewController
:(UIPageViewController *)pageViewController
spineLocationForInterfaceOrientation:(UIInterfaceOrientation)orientation;
```

Первый метод вызывается, когда пользователь переходит к следующей или предыдущей странице *или* решает перелистнуть страницу вперед или назад, но передумывает в момент, пока страница еще движется. (В последнем случае пользователь возвращается к той странице, которую просматривал перед актом листания.) Свойство `transitionCompleted` получает значение `YES`, если удалось отобразить анимацию листания страницы, и `NO` — если пользователь решил страницу не перелистывать и прервал анимацию в ходе ее выполнения.

Второй метод вызывается при каждом изменении ориентации устройства. Этот метод можно использовать для того, чтобы указывать положение сгиба страницы, возвращая значение типа `UIPageViewControllerSpineLocation`:

```
typedef NS_ENUM(NSInteger, UIPageViewControllerSpineLocation) {
  UIPageViewControllerSpineLocationNone = 0,
  UIPageViewControllerSpineLocationMin = 1,
  UIPageViewControllerSpineLocationMid = 2,
  UIPageViewControllerSpineLocationMax = 3
};
```

Возможно, все это выглядит немного запутанно, но позвольте мне продемонстрировать, что имеется в виду. Если мы используем расположение сгиба `UIPageViewControllerSpineLocationMin`, то для отображения страничного вида пользователю потребуется всего один контроллер вида. Если пользователь перейдет к следующей странице, то увидит уже новый контроллер вида. Но если мы зададим для отображения сгиба `UIPageViewControllerSpineLocationMid`, то для демонстрации такого варианта нам понадобятся уже два контроллера видов одновременно. Один будет представлять левую страницу, другой — правую, а между ними расположится сгиб. Сейчас покажу, что я имею в виду. На рис. 1.76 изображен пример страничного вида, имеющего альбомную ориентацию. Здесь для расположения изгиба выбрано значение `UIPageViewControllerSpineLocationMin`.

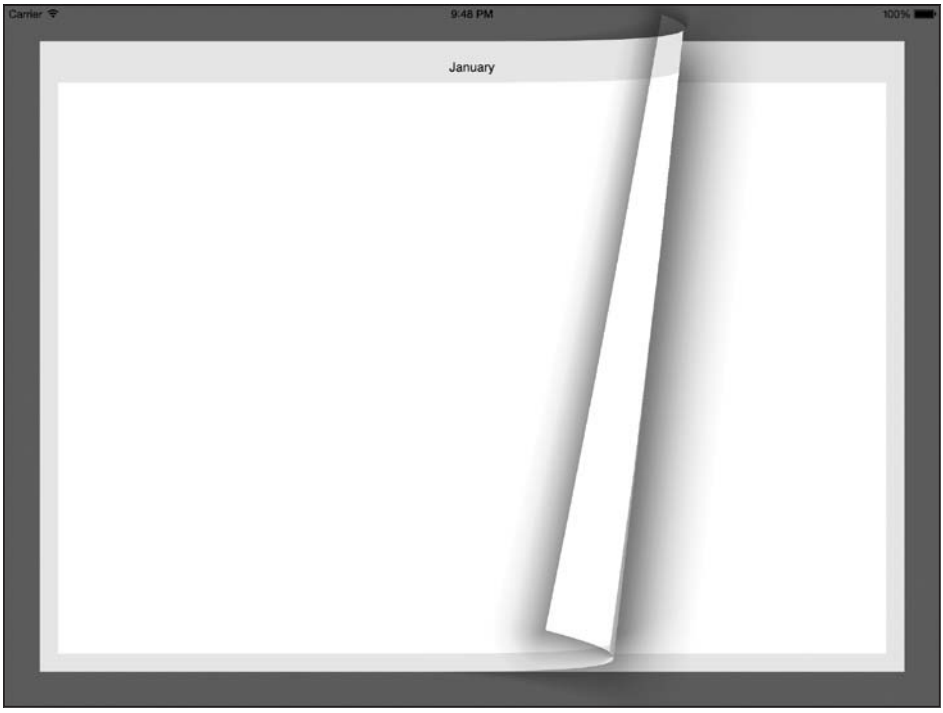


**Рис. 1.76.** Один контроллер вида. Представлен контроллер вида-страницы с альбомной ориентацией

Теперь, если вернуть расположение сгиба, соответствующее `UIPageViewControllerSpineLocationMid`, получим примерно такой результат, как на рис. 1.77.

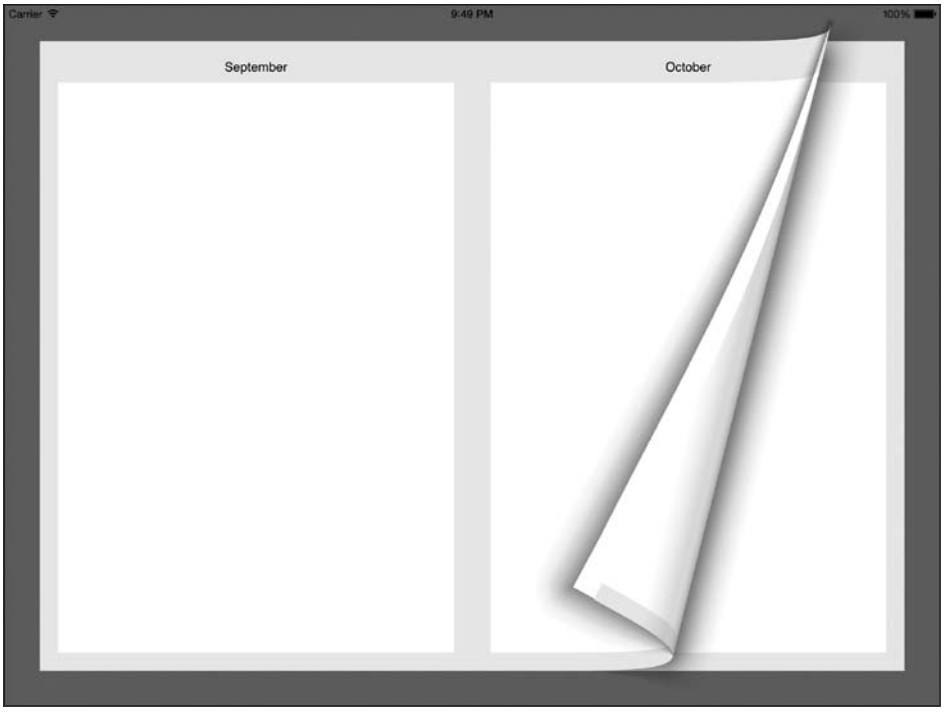


**Рис. 1.77.** Два контроллера видов, отображенные в контроллере вида-страницы, где страница имеет альбомную ориентацию

Как видно на рис. 1.77, сгиб расположен точно по центру экрана, между двумя контроллерами видов. Когда пользователь перелистывает страницу справа налево, страница оказывается слева, а справа контроллер вида-страницы отображает новую страницу. Вся логика заключена в следующем методе делегата:

```
- (UIPageViewControllerSpineLocation)pageViewController
:(UIPageViewController *)pageViewController
spineLocationForInterfaceOrientation:(UIInterfaceOrientation)orientation;
```

Итак, мы разобрались с делегатом контроллера страничного вида, а что насчет источника данных? Источник данных контроллера страничного вида должен соответствовать протоколу `UIPageViewControllerDataSource`. Этот протокол предоставляет два следующих важных метода:

```
- (UIViewController *)
  pageViewController:(UIPageViewController *)pageViewController
  viewControllerBeforeViewController:(UIViewController *)viewController;

- (UIViewController *)
pageViewController:(UIPageViewController *)pageViewController
viewControllerAfterViewController:(UIViewController *)viewController;
```

Первый метод вызывается, когда контроллер вида-страницы уже имеет на экране устройства один контроллер вида и должен узнать, какой из предыдущих кон-

троллеров видов нужно отображать. Это происходит, когда пользователь решает перейти на следующую страницу (перелистнуть имеющуюся). Второй метод вызывается, когда контроллеру вида необходимо узнать, какой контроллер вида отобразить вслед за той страницей, которую пользователь перелистнет.

Как вы могли убедиться, среда Xcode значительно упрощает создание приложений с постраничной организацией. Все, что, по сути, от вас требуется, — предоставить содержимое для модели данных (`ModelController`) и двигаться дальше. Если требуется отдельно настроить цвета и изображения в контроллерах ваших видов, то можно либо сделать это в конструкторе интерфейса (Interface Builder), позволяющем напрямую изменять файлы раскадровки, либо написать собственный код для реализации каждого из контроллеров видов.

# 1.29. Отображение вспомогательных экранов с помощью UIPopoverController

## Постановка задачи

Вы хотите отображать на iPad окно с информацией, не занимая при этом целый экран.

## Решение

Воспользуйтесь вспомогательными экранами.

## Обсуждение

Вспомогательные экраны (Popover) применяются для вывода на экран iPad дополнительной информации. В качестве примера можно привести браузер Safari из iPad. Если пользователь нажмет кнопку Bookmarks (Закладки), то на экране появится еще одно окошко, в котором будет перечислено содержимое панели закладок (рис. 1.78).

По умолчанию, если на устройстве отображен вспомогательный экран, а пользователь нажмет что-нибудь за его пределами, этот вспомогательный экран автоматически закроется. Вы можете задать поведение, при котором вспомогательный экран не закрывается, когда пользователь дотрагивается до какой-то конкретной части экрана, — об этом поговорим в дальнейшем. Содержимое вспомогательных экранов отображается с применением контроллеров видов. Обратите внимание: на вспомогательных экранах могут присутствовать и навигационные контроллеры, поскольку они являются подклассами `UIViewController`.

Вспомогательные экраны применяются только на устройствах iPad. Если у вас есть контроллер вида, чей код выполняется и на iPad, и на iPhone, необходимо гарантировать, что вспомогательные экраны не будут инстанцироваться на других устройствах, кроме iPad.

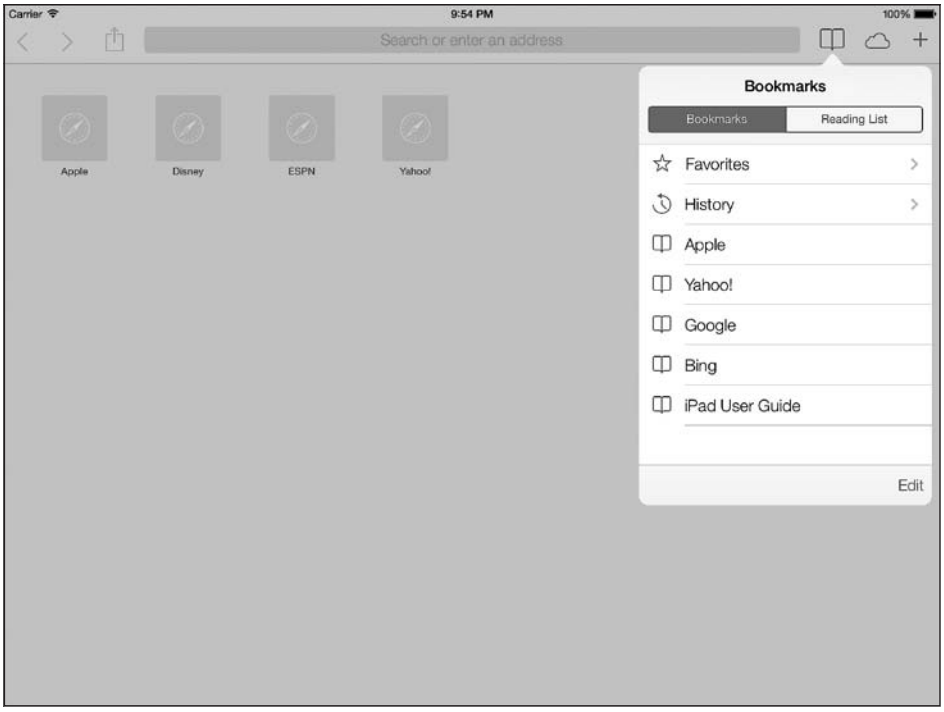


**Рис. 1.78.** Вспомогательный экран с закладками браузера Safari на планшете iPad

Вспомогательные экраны можно отображать и использовать двумя способами:

- открывать их из навигационной кнопки экземпляра `UIBarButtonItem`;
- открывать из прямоугольной области в виде.

При изменении ориентации (то есть при повороте) устройства вспомогательные экраны либо закрываются, либо временно скрываются. Следует позаботиться о том, чтобы пользователю было удобно работать с программой, вновь отобразив вспомогательный экран, когда устройство будет окончательно переориентировано, — если это возможно. В определенных случаях вспомогательный экран можно закрывать автоматически после того, как ориентация устройства изменится. Например, если пользователь нажимает какую-либо навигационную кнопку, когда устройство находится в альбомном режиме, вы можете отобразить вспомогательный экран. Предположим, что ваше приложение сделано так, что, когда ориентация изменяется на книжную, данная навигационная кнопка по какой-то причине удаляется с панели. Теперь, чтобы не путать пользователя, после изменения ориентации устройства на книжную нужно убрать и вспомогательный экран, ассоциированный с этой кнопкой. Но в некоторых случаях приходится немного импровизировать со вспомогательными экранами, чтобы пользователю было удобнее работать с программой, так как управление ориентацией устройства — более тонкий процесс, чем можно предположить из предыдущего сценария.

Для создания демонстрационного приложения со вспомогательными экранами нужно сначала обдумать стратегию, зависящую от поставленных перед вами требований. Например, требуется написать приложение с контроллером вида, загруженным в навигационный контроллер. В корневом контроллере вида будет отображаться кнопка +, расположенная в правом углу навигационной панели. Если на устройстве iPad нажать кнопку +, откроется вспомогательный экран с двумя кнопками. На одной будет написано Photo (Фото), на другой — Audio (Аудио). При нажатии той же самой навигационной кнопки на iPhone отобразится предупреждающий вид (Alert View) с тремя кнопками — двумя вышеупомянутыми и кнопкой Cancel (Отмена), чтобы пользователь мог сам закрыть это окно, если захочет. При нажатии любой из этих кнопок (и в предупреждающем виде iPhone, и на вспомогательном экране iPad) мы фактически ничего не сделаем — просто уберем это окошко.

Итак, продолжим и создадим в Xcode универсальный проект Single View (Приложение с единственным видом). Назовем проект Displaying Popovers with UIPopoverController («Отображение вспомогательных экранов с помощью UIPopoverController»). Затем, воспользовавшись приемами, описанными в разделе 6.1, добавим в раскадровку навигационный контроллер, чтобы у контроллеров видов появилась навигационная панель.

После этого перейдем к определению корневого контроллера вида и укажем здесь свойство типа `UIPopoverController`:

```
#import "ViewController.h"

@interface ViewController () <UIAlertViewDelegate>
@property (nonatomic, strong) UIPopoverController *myPopoverController;
@property (nonatomic, strong) UIBarButtonItem *barButtonAdd;
@end

@implementation ViewController

<# Оставшаяся часть вашего кода находится здесь #>
```

Как видите, мы также определяем для контроллера вида свойство `barButtonAdd`. Это навигационная кнопка, которую мы добавим на нашу панель. Мы собираемся отображать вспомогательный экран после того, как пользователь нажмет эту кнопку (подробнее о навигационных кнопках рассказано в разделе 1.15). При этом необходимо гарантировать, что мы инстанцируем вспомогательный экран только для iPad. Прежде чем идти дальше и инстанцировать корневой контроллер вида с навигационной кнопкой, создадим подкласс от `UIViewController` и назовем его `PopoverContentViewController`. В дальнейшем будем отображать его содержимое на вспомогательном экране. В разделе 1.9 подробнее рассказано о контроллерах видов и о том, как их создавать.

В контроллере информационного вида, отображаемого на вспомогательном экране, будет две кнопки (как мы и рассчитывали). Тем не менее в этом контроллере вида должна быть также ссылка на контроллер вспомогательного экрана. Она нужна, чтобы убрать вспомогательный экран, как только пользователь нажмет любую из кнопок. Сначала в контроллере информационного вида нужно определить специальное свойство для ссылки на вспомогательный экран:

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

@interface PopoverContentViewController : UIViewController

/* Не следует определять данное свойство как strong. В противном случае
 возникнет цикл удержания (Retain Cycle) между контроллером
 информационного вида и контроллером вспомогательного экрана,
 так как контроллер вспомогательного экрана не даст исчезнуть
 контроллеру информационного вида и наоборот. */
@property (nonatomic, weak) UIPopoverController *popoverController;

@end
```

И здесь же, в файле реализации контроллера вида с содержимым, объявим кнопки панели:

```
#import "PopoverContentViewController.h"

@interface PopoverContentViewController ()
@property (nonatomic, strong) UIButton *buttonPhoto;
@property (nonatomic, strong) UIButton *buttonAudio;
@end

@implementation PopoverContentViewController

<# Оставшаяся часть вашего кода находится здесь #>
```

Затем создадим две кнопки в контроллере информационного вида и свяжем их ссылками с методами, обеспечивающими их функционирование. Эти методы будут закрывать тот вспомогательный экран, в котором отображается контроллер информационного вида. Не забывайте, что контроллер вспомогательного экрана будет присваивать себя свойству `popoverController`, относящемуся к контроллеру информационного вида:

```
- (BOOL) isInPopover{

  Class popoverClass = NSClassFromString(@"UIPopoverController");

  if (popoverClass != nil &&
      UI_USER_INTERFACE_IDIOM() == UIUserInterfaceIdiomPad &&
      self.popoverController != nil){
      return YES;
  } else {
    return NO;
  }

}

- (void) gotoAppleWebsite:(id)paramSender{

  if ([self isInPopover]){
    /* Перейти на сайт и закрыть вспомогательный экран. */
    [self.popoverController dismissPopoverAnimated:YES];
```

```
  } else {
    /* Обработать ситуацию с iPhone. */
  }

}

- (void) gotoAppleStoreWebsite:(id)paramSender{

  if ([self isInPopover]){
    /* Перейти на сайт и закрыть вспомогательный экран. */
    [self.popoverController dismissPopoverAnimated:YES];
  } else {
    /* Обработать ситуацию с iPhone. */
  }

}

- (void)viewDidLoad{
  [super viewDidLoad];

  self.contentSizeForViewInPopover = CGSizeMake(200.0f, 125.0f);

  CGRect buttonRect = CGRectMake(20.0f,
                                 20.0f,
                                 160.0f,
                                 37.0f);

  self.buttonPhoto = [UIButton buttonWithType:UIButtonTypeRoundedRect];
  [self.buttonPhoto setTitle:@"Photo"
                    forState:UIControlStateNormal];
  [self.buttonPhoto addTarget:self
                       action:@selector(gotoAppleWebsite:)
             forControlEvents:UIControlEventTouchUpInside];

  self.buttonPhoto.frame = buttonRect;

  [self.view addSubview:self.buttonPhoto];

  buttonRect.origin.y += 50.0f;
  self.buttonAudio = [UIButton buttonWithType:UIButtonTypeRoundedRect];

  [self.buttonAudio setTitle:@"Audio"
                    forState:UIControlStateNormal];
  [self.buttonAudio addTarget:self
                       action:@selector(gotoAppleStoreWebsite:)
             forControlEvents:UIControlEventTouchUpInside];

  self.buttonAudio.frame = buttonRect;

  [self.view addSubview:self.buttonAudio];

}
```

Теперь в методе viewDidLoad корневого контроллера вида создадим навигационную кнопку. В зависимости от типа устройства при нажатии навигационной кнопки мы будем отображать либо вспомогательный экран (на iPad), либо предупреждающий вид (на iPhone):

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

  /* Проверяем, существует ли этот класс в том варианте iOS,
  где действует приложение. */
  Class popoverClass = NSClassFromString(@"UIPopoverController");

  if (popoverClass != nil &&
      UI_USER_INTERFACE_IDIOM() == UIUserInterfaceIdiomPad){

      PopoverContentViewController *content =
      [[PopoverContentViewController alloc] initWithNibName:nil
                                                      bundle:nil];

      self.popoverController = [[UIPopoverController alloc]
      initWithContentViewController:content];

      content.popoverController = self.popoverController;

      self.barButtonAdd = [[UIBarButtonItem alloc]
      initWithBarButtonSystemItem:UIBarButtonSystemItemAdd
      target:self
      action:@selector(performAddWithPopover:)];

  } else {

      self.barButtonAdd = [[UIBarButtonItem alloc]
      initWithBarButtonSystemItem:UIBarButtonSystemItemAdd
      target:self
      action:@selector(performAddWithAlertView:)];

  }

  [self.navigationItem setRightBarButtonItem:self.barButtonAdd
                                    animated:NO];

}
```

Контроллер вспомогательного экрана ставит на себя ссылку в контроллере информационного вида сразу после инициализации информационного вида. Это очень важно. Контроллер вспомогательного экрана невозможно инициализировать в отсутствие контроллера информационного вида. Как только контроллер вспомогательного экрана инициализирован посредством контроллера информационного вида, можно продолжать работу и изменять контроллер информационного вида в контроллере вспомогательного экрана — но этого нельзя делать в процессе инициализации.

Мы решили, что при нажатии навигационной кнопки + на устройстве iPad будет запускаться метод `performAddWithPopover:`. Если мы имеем дело не с iPad, то нужно, чтобы при нажатии этой кнопки запускался метод `performAddWithAlertView:`. Итак, реализуем два этих метода, а также позаботимся о методах делегатов предупреждающего вида — чтобы нам было известно, какую кнопку в предупреждающем виде нажимает пользователь, работающий с iPhone:

```
- (NSString *) photoButtonTitle{
  return @"Photo";
}

- (NSString *) audioButtonTitle{
  return @"Audio";
}

- (void) alertView:(UIAlertView *)alertView
  didDismissWithButtonIndex:(NSInteger)buttonIndex{

  NSString *buttonTitle = [alertView buttonTitleAtIndex:buttonIndex];

  if ([buttonTitle isEqualToString:[self photoButtonTitle]]){
    /* Добавляем фотографию... */
  }
  else if ([buttonTitle isEqualToString:[self audioButtonTitle]]){
   /* Добавляем аудио... */
  }

}

- (void) performAddWithAlertView:(id)paramSender{

  [[[UIAlertView alloc] initWithTitle:nil
                              message:@"Add..."
                             delegate:self
                    cancelButtonTitle:@"Cancel"
                    otherButtonTitles:
    [self photoButtonTitle],
    [self audioButtonTitle], nil] show];

}

- (void) performAddWithPopover:(id)paramSender{
  [self.popoverController
  presentPopoverFromBarButtonItem:self.barButtonAdd
  permittedArrowDirections:UIPopoverArrowDirectionAny
  animated:YES];

}
```

Если запустить это приложение в эмуляторе iPad, то при нажатии кнопки + на навигационной панели мы увидим примерно такой интерфейс, как на рис. 1.79.


**Рис. 1.79.** Простой вспомогательный экран, отображаемый после нажатия навигационной кнопки

Если запустить это же универсальное приложение в эмуляторе iPhone и нажать на навигационной панели кнопку +, результат будет примерно как на рис. 1.80.

Здесь мы воспользовались важным свойством контроллера информационного вида: `preferredContentSize`. Когда вспомогательный экран отображает контроллер своего информационного вида, он будет автоматически считывать значение этого свойства и корректировать свой размер (высоту и ширину). Кроме того, мы использовали метод `presentPopoverFromBarButtonItem:permittedArrowDirections:animated:` вспомогательного экрана в корневом контроллере нашего вида. Этот метод нужен, чтобы вспомогательный экран отображался над кнопкой навигационной панели. Первый параметр, принимаемый данным методом, — это кнопка навигационной панели, та, над которой должен всплывать контроллер вспомогательного экрана. Второй параметр указывает при появлении вспомогательного экрана направление его развертывания относительно объекта, из которого он появляется. Например, на рис. 1.79 видно, что стрелка вспомогательного экрана указывает вверх от кнопки с навигационной панели. Значение, передаваемое этому параметру, должно относиться к типу `UIPopoverArrowDirection::`

```
typedef NS_OPTIONS(NSUInteger, UIPopoverArrowDirection) {
  UIPopoverArrowDirectionUp = 1UL << 0,
  UIPopoverArrowDirectionDown = 1UL << 1,
  UIPopoverArrowDirectionLeft = 1UL << 2,
```


**Рис. 1.80.** В универсальном приложении вспомогательные экраны заменяются предупреждающими видами

```
  UIPopoverArrowDirectionRight = 1UL << 3,
  UIPopoverArrowDirectionAny = UIPopoverArrowDirectionUp |
  UIPopoverArrowDirectionDown |
  UIPopoverArrowDirectionLeft |
  UIPopoverArrowDirectionRight,
  UIPopoverArrowDirectionUnknown =
  NSUIntegerMax
};
```

## См. также

Разделы 1.9 и 1.15.

# 2 Создание динамических и интерактивных пользовательских интерфейсов

## 2.0. Введение

Когда iPhone только появился на рынке, он поистине задал стандарт интерактивности в мобильных приложениях. Приложения iOS были и остаются поразительно интерактивными — вы можете на ходу манипулировать различными компонентами пользовательского интерфейса, корректируя их для максимально полного удовлетворения своих потребностей.

В iOS 7 Apple добавила в iOS SDK ряд новых классов, которые позволяют обогащать ваше приложение очень интересной физикой, делая их еще более интерактивными. Например, в новой iOS вы заметите, что фоновые рисунки, которые могут служить обоями Рабочего стола, стали еще живее, так как они могут двигаться по экрану, если вы качаете устройство влево-вправо, и т. д. Появились также новые разновидности поведений, которые iOS позволяет добавлять в приложения.

Приведу другой пример. Допустим, у вас есть приложение для обмена фотографиями, работающее на iPad. В левой части экрана находятся несколько картинок, которые были извлечены из пользовательского фотоальбома на Рабочий стол. Справа расположен компонент, напоминающий корзину. Каждая фотография, перенесенная в корзину, будет доступна для пакетного совместного использования через какую-нибудь социальную сеть, например Facebook. Вы хотите обогатить интерактивность приложения с помощью анимации так, чтобы пользователь мог «кидать» фотографии в корзину слева, а фотографии закреплялись в корзине. Все это можно было сделать и раньше, но для выполнения таких операций требовались глубокие знания Core Animation, а также довольно хорошее понимание физики.

Пользуясь UI Dynamics — новой технологией от Apple, — вы можете значительно упростить реализацию многих таких возможностей в приложении. На самом

деле достаточно всего нескольких строк кода, чтобы реализовать в ваших видах очень интересную физику и необычное поведение.

Apple категоризировала такие действия в классах *поведений*, которые можно прикреплять к *аниматору*. Например, вы можете добавить к кнопке в вашем виде *тяготение*. В таком случае кнопка будет падать из верхней части вида (если вы ее там поместили) до самого его низа и даже сможет выпадать за его пределы. Если вы хотите этому воспрепятствовать и сделать так, чтобы кнопка падала только до дна вида и не дальше, то к аниматору нужно прикрепить и другое поведение — *столкновение*. Аниматор будет управлять поведениями, которые вы добавите к разным видам вашего приложения, а также их взаимодействиями. Вам не придется об этом беспокоиться. Далее перечислены несколько классов, обеспечивающих различное поведение компонентов пользовательского интерфейса:

- `UICollisionBehavior` — обеспечивает обнаружение столкновений;
- `UIGravityBehavior` — как понятно из названия, обеспечивает имитацию тяготения;
- `UIPushBehavior` — позволяет имитировать в ваших видах толчки. Допустим, вы дотронулись пальцем до экрана, а потом стали постепенно двигать палец к его верхнему краю. Если к виду прикреплена кнопка, оснащенная толчковым поведением, то вы могли бы толкать эту кнопку пальцем, как если бы она лежала на столе;
- `UISnapBehavior` — обеспечивает прикрепление видов к тем или иным местам на экране.

Как было указано ранее, для каждого динамического поведения потребуется аниматор типа `UIDynamicAnimator`. Аниматор должен быть инициализирован с сущностью, которая в Apple называется *опорным видом*. Аниматор использует систему координат опорного вида для расчета результата различных поведений. Например, можно указать вид определенного контроллера вида в качестве опорного для динамического аниматора. В таком случае можно добавить к аниматору поведение столкновения и тем самым гарантировать, что добавляемые элементы не будут выпадать за пределы опорного вида. Таким образом вы сможете разместить в опорном виде все элементы пользовательского интерфейса, даже если на них действует поведение тяготения.

Опорный вид используется также в качестве контекста для анимации, которой управляет аниматор. Например, аниматору требуется определить, столкнутся ли два квадрата друг с другом. Для этого он использует методы Core Graphics, позволяющие определить, будут ли два этих квадрата накладываться друг на друга в контексте их вышестоящего вида — в данном случае опорного вида.

В этой главе мы исследуем различные комбинации подобных поведений и поговорим о том, как вы сможете сделать свои приложения более интерактивными, вооружившись поведениями UIKit и аниматорами. Начнем с простых примеров и постепенно будем выстраивать на их основе изучаемый материал, знакомясь с все более захватывающими примерами.

# 2.1. Добавление тяготения к компонентам пользовательского интерфейса

## Постановка задачи

Необходимо, чтобы компоненты вашего пользовательского интерфейса двигались так, как будто на них действует сила тяжести: например, если перетащить элемент к верхнему краю экрана, то под действием силы тяжести он упадет к нижнему краю. Объединив эту возможность с поведением столкновения, которое мы изучим позднее, можно создавать такие компоненты пользовательского интерфейса, которые будут падать со своего действительного местоположения, пока не столкнутся с указанной вами линией.

## Решение

Инициализируйте объект типа `UIGravityBehavior` и добавьте к нему те компоненты пользовательского интерфейса, которые должны испытывать тяготение к этому объекту. Сделав это, создайте экземпляр `UIDynamicAnimator`, добавьте к аниматору поведение тяготения, а всю остальную работу аниматор сделает за вас.

## Обсуждение

В этом разделе мы создадим простой вид, представляющий собой раскрашенный квадрат, находящийся в одновидовом приложении. Этот вид мы поместим в центре экрана. Затем добавим к нему поведение тяготения и посмотрим, как он будет падать вниз, пока не скроется за пределами экрана.

Итак, определим аниматор и вид:

```
#import "ViewController.h"

@interface ViewController ()
@property (nonatomic, strong) UIView *squareView;
@property (nonatomic, strong) UIDynamicAnimator *animator;
@end

@implementation ViewController

<# Оставшаяся часть кода вашего контроллера вида находится здесь #>
```

Далее создадим небольшой вид, присвоим ему цвет и поместим в центре вида нашего контроллера. Так мы получим экземпляр класса `UIGravityBehavior`, воспользовавшись методом-инициализатором `initWithItems:`. Этот инициализатор принимает массив объектов, соответствующих протоколу `UIDynamicItem`. По умолчанию этому протоколу соответствуют все экземпляры `UIView`, поэтому, как только вид готов, можно идти дальше:

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

```
    /* Создаем маленький квадратный вид и добавляем его к self.view */
    self.squareView = [[UIView alloc] initWithFrame:
                       CGRectMake(0.0f, 0.0f, 100.0f, 100.0f)];
    self.squareView.backgroundColor = [UIColor greenColor];
    self.squareView.center = self.view.center;
    [self.view addSubview:self.squareView];

    /* Создаем аниматор и реализуем тяготение */
    self.animator = [[UIDynamicAnimator alloc]
                    initWithReferenceView:self.view];

    UIGravityBehavior *gravity = [[UIGravityBehavior alloc]
                                  initWithItems:@[self.squareView]];

    [self.animator addBehavior:gravity];

}
```

Если вы не хотите добавлять тяготение ко всем вашим видам, как только инициализируете это поведение, то можете добавить его позже с помощью метода экземпляра addItem:, относящегося к классу UIGravityBehavior. Этот метод также принимает объект, соответствующий указанному ранее протоколу.

Теперь запустите ваше приложение. Как только вид в контроллере появится на экране, вы увидите цветной квадрат, падающий из центра экрана вниз, до нижнего края, а потом скрывающийся за пределами дисплея. Поведение тяготения, точно как реальная сила тяжести, заставляет элементы двигаться вниз, вплоть до определенной границы. Поскольку в данном случае никакой границы нет, элемент падает вниз до бесконечности. Мы исправим этот недостаток позже в данной главе, реализовав для элементов поведение столкновения.

## См. также

Раздел 2.0.

# 2.2. Обнаружение столкновений между компонентами пользовательского интерфейса и реагирование на них

## Постановка задачи

Требуется задать на экране границы столкновений между компонентами вашего пользовательского интерфейса так, чтобы эти компоненты не перекрывали друг друга.

## Решение

Инстанцируйте объект типа `UICollisionBehavior` и прикрепите его к объекту аниматора. Присвойте свойству `translatesReferenceBoundsIntoBoundary` поведения столкновений значение `YES` и убедитесь в том, что аниматор инициализирован с вышестоящим видом в качестве опорной сущности. Так вы гарантируете, что дочерние виды, на которые распространяется поведение столкновения (о чем мы вскоре поговорим), не будут выпадать за пределы вышестоящего вида.

## Обсуждение

Поведение столкновения, относящееся к типу `UICollisionBehavior`, затрагивает объекты, соответствующие протоколу `UIDynamicItem`. Все виды типа `UIView` уже ему соответствуют, поэтому вам придется лишь инстанцировать ваши виды и добавить их к поведению столкновения. Поведение столкновения требует определить на экране границы, которые будут непреодолимы для элементов, находящихся в аниматоре. Например, если вы зададите линию, которая будет идти из нижнего левого угла вашего опорного вида в нижний правый угол (соответственно, это будет линия, вплотную прилегающая к его нижнему краю), а также добавите к этому виду поведение тяготения, то виды, расположенные на экране, будут двигаться под действием тяготения вниз, но не смогут «провалиться» с экрана, так как столкнутся с его нижним краем, который задается поведением столкновения.

Если вы хотите, чтобы границы области, в которой действует поведение столкновения, совпадали с границами опорного вида, то присвойте свойству `translatesReferenceBoundsIntoBoundary` экземпляра поведения столкновения значение `YES`. Если хотите самостоятельно провести линии, соответствующие границам такой области, просто воспользуйтесь методом экземпляра `addBoundaryWithIdentifier:fromPoint:toPoint:`, относящимся к классу `UICollisionBehavior`.

В этом примере мы собираемся создать два цветных вида, один из которых расположен на другом. После этого добавим к аниматору поведение тяготения, чтобы эти виды не перекрывали друг друга. Кроме того, они не будут выходить за границы опорного вида (то есть вида контроллера).

Итак, для начала определим массив видов и аниматор:

```
#import "ViewController.h"

@interface ViewController ()
@property (nonatomic, strong) NSMutableArray *squareViews;
@property (nonatomic, strong) UIDynamicAnimator *animator;
@end
@implementation ViewController

<# Остаток вашего кода находится здесь #>
```

Потом, когда вид появится на экране, зададим поведения столкновения и тяготения и добавим их к аниматору:

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

  /* Создаем виды */
  NSUInteger const NumberOfViews = 2;

  self.squareViews = [[NSMutableArray alloc] initWithCapacity:NumberOfViews];
  NSArray *colors = @[[UIColor redColor], [UIColor greenColor]];

  CGPoint currentCenterPoint = self.view.center;
  CGSize eachViewSize = CGSizeMake(50.0f, 50.0f);
  for (NSUInteger counter = 0; counter < NumberOfViews; counter++){

    UIView *newView =
    [[UIView alloc] initWithFrame:
     CGRectMake(0.0f, 0.0f, eachViewSize.width, eachViewSize.height)];

    newView.backgroundColor = colors[counter];
    newView.center = currentCenterPoint;

    currentCenterPoint.y += eachViewSize.height + 10.0f;

    [self.view addSubview:newView];

    [self.squareViews addObject:newView];

  }

  self.animator = [[UIDynamicAnimator alloc]
                   initWithReferenceView:self.view];

  /* Создаем тяготение */
  UIGravityBehavior *gravity = [[UIGravityBehavior alloc]
                                initWithItems:self.squareViews];
  [self.animator addBehavior:gravity];

  /* Реализуем обнаружение столкновений */
  UICollisionBehavior *collision = [[UICollisionBehavior alloc]
                                    initWithItems:self.squareViews];
  collision.translatesReferenceBoundsIntoBoundary = YES;
  [self.animator addBehavior:collision];
}
```

Получим примерно такой же результат, как на рис. 2.1.

В этом примере показано, что поведение обнаружения столкновений работает отлично, если свойство `translatesReferenceBoundsIntoBoundary` имеет значение `YES`. Но что, если мы захотим очертить собственные границы столкновений? Здесь и пригодится метод экземпляра `addBoundaryWithIdentifier:fromPoint:toPoint:`, относящийся к поведению столкновения. Вот параметры, которые следует передать этому методу:

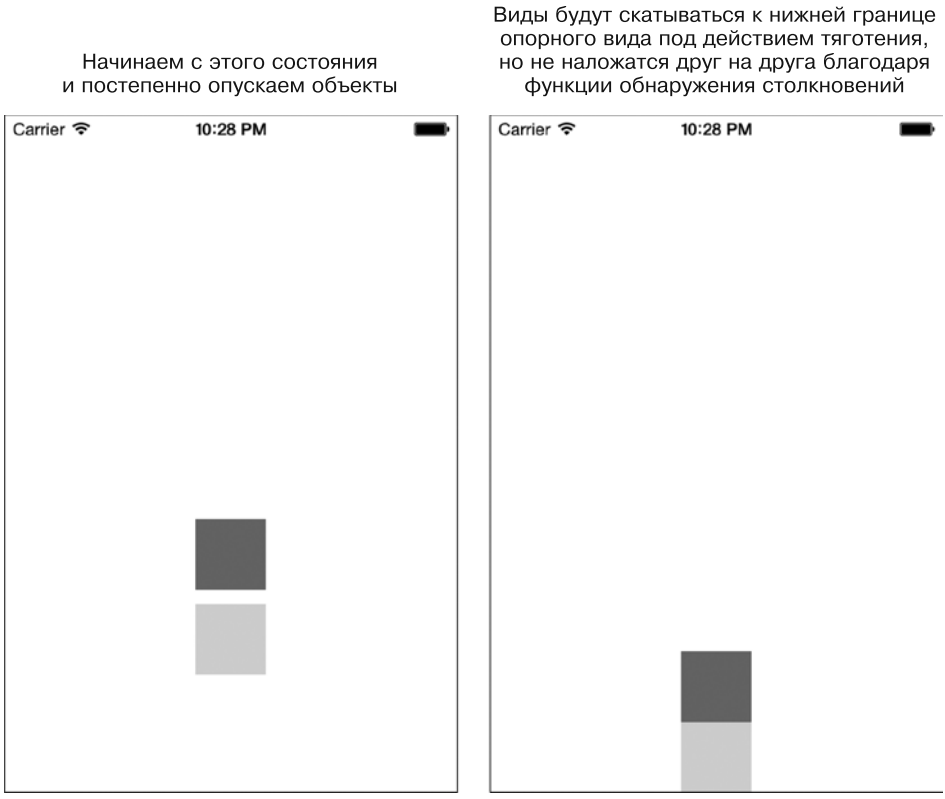


**Рис. 2.1.** Взаимодействующие поведения тяготения и столкновения

- `addBoundaryWithIdentifier` — строковый идентификатор для вашей границы. Он используется для того, чтобы впоследствии вы могли получить от границы информацию о столкновении. Вы могли бы передать такой же идентификатор методу `boundaryWithIdentifier:` и получить в ответ объект границы. Объект относится к типу `UIBezierPath` и может поддерживать довольно сложные, сильно искривленные границы. Но большинство программистов предпочитают указывать простые горизонтальные или вертикальные границы, что мы и сделаем;
- `fromPoint` — начальная точка границы, относится к типу `CGPoint`;
- `toPoint` — конечная точка границы, относится к типу `CGPoint`.

Итак, предположим, что вы хотите провести границу в нижней части опорного вида (в данном случае вида с контроллером), но не хотите, чтобы она совпадала с нижним краем. Вместо этого вам нужна граница, расположенная на 100 точек выше нижнего края. В таком случае свойство поведения столкновения `translatesReferenceBoundsIntoBoundary` не поможет, так как вы хотите задать иную границу, не совпадающую с пределами опорного вида. Нужно воспользоваться методом `addBoundaryWithIdentifier:fromPoint:toPoint:`, вот так:

```
/* Создаем обнаружение столкновений */
UICollisionBehavior *collision = [[UICollisionBehavior alloc]
                                  initWithItems:self.squareViews];
[collision
  addBoundaryWithIdentifier:@"bottomBoundary"
  fromPoint:CGPointMake(0.0f, self.view.bounds.size.height – 100.0f)
  toPoint:CGPointMake(self.view.bounds.size.width,
                      self.view.bounds.size.height – 100.0f)];

[self.animator addBehavior:collision];
```

Теперь, если мы объединим это поведение с тяготением, как делали раньше, то квадраты будут падать в опорном виде сверху вниз, но не достигнут его дна, так как проведенная нами нижняя граница находится немного выше. В рамках этого раздела я также хочу продемонстрировать возможность обнаружения столкновений между различными элементами, обладающими поведением столкновения. Класс `UICollisionBehavior` имеет свойство `collisionDelegate`, которое будет выступать в качестве делегата при обнаружении столкновений у элементов, обладающих поведением столкновения. Этот объект-делегат должен соответствовать протоколу `UICollisionBehaviorDelegate`. Данный протокол обладает некоторыми методами, которые мы можем реализовать. Вот два наиболее важных из этих методов:

- `collisionBehavior:beganContactForItem:withBoundaryIdentifier:atPoint:` — вызывается в делегате, когда один из элементов, обладающих поведением столкновения, ударяется об одну из границ, добавленных к этому поведению;
- `collisionBehavior:endedContactForItem:withBoundaryIdentifier:atPoint:` — вызывается, когда элемент, столкнувшийся с границей, отскочил от нее и, таким образом, контакт элемента с границей прекратился.

Чтобы продемонстрировать вам делегат в действии и показать, как его можно использовать, расширим приведенный пример. Как только квадратики достигают нижней границы опорного вида, мы делаем их красными, увеличиваем на 200 %, а потом заставляем рассыпаться, как при взрыве, и исчезать из виду:

```
NSString *const kBottomBoundary = @"bottomBoundary";

@interface ViewController () <UICollisionBehaviorDelegate>
@property (nonatomic, strong) NSMutableArray *squareViews;
@property (nonatomic, strong) UIDynamicAnimator *animator;
@end

@implementation ViewController

- (void)collisionBehavior:(UICollisionBehavior*)paramBehavior
      beganContactForItem:(id <UIDynamicItem>)paramItem
   withBoundaryIdentifier:(id <NSCopying>)paramIdentifier
                  atPoint:(CGPoint)paramPoint{

  NSString *identifier = (NSString *)paramIdentifier;

  if ([identifier isEqualToString:kBottomBoundary]){
```

```
    [UIView animateWithDuration:1.0f animations:^{
      UIView *view = (UIView *)paramItem;

      view.backgroundColor = [UIColor redColor];
      view.alpha = 0.0f;
      view.transform = CGAffineTransformMakeScale(2.0f, 2.0f);
    } completion:^(BOOL finished) {
      UIView *view = (UIView *)paramItem;
      [paramBehavior removeItem:paramItem];
      [view removeFromSuperview];
    }];

  }

}

- (void)viewDidAppear□BOOL)animated{
  [super viewDidAppear:animated];

  /* Создаем виды */
  NSUInteger const NumberOfViews = 2;

  self.squareViews = [[NSMutableArray alloc] initWithCapacity:NumberOfViews];
  NSArray *colors = @[[UIColor redColor], [UIColor greenColor]];

  CGPoint currentCenterPoint = CGPointMake(self.view.center.x, 0.0f);
  CGSize eachViewSize = CGSizeMake(50.0f, 50.0f);
  for (NSUInteger counter = 0; counter < NumberOfViews; counter++){

    UIView *newView =
    [[UIView alloc] initWithFrame:
     CGRectMake(0.0f, 0.0f, eachViewSize.width, eachViewSize.height)];

    newView.backgroundColor = colors[counter];
    newView.center = currentCenterPoint;

    currentCenterPoint.y += eachViewSize.height + 10.0f;
    [self.view addSubview:newView];

    [self.squareViews addObject:newView];

  }

  self.animator = [[UIDynamicAnimator alloc]
                   initWithReferenceView:self.view];

  /* Создаем тяготение */
  UIGravityBehavior *gravity = [[UIGravityBehavior alloc]
                                initWithItems:self.squareViews];
  [self.animator addBehavior:gravity];
```

```
  /* Создаем обнаружение столкновений */
  UICollisionBehavior *collision = [[UICollisionBehavior alloc]
                                    initWithItems:self.squareViews];
  [collision
    addBoundaryWithIdentifier:kBottomBoundary
    fromPoint:CGPointMake(0.0f, self.view.bounds.size.height - 100.0f)
    toPoint:CGPointMake(self.view.bounds.size.width,
                        self.view.bounds.size.height - 100.0f)];
  collision.collisionDelegate = self;

  [self.animator addBehavior:collision];

}
```

Объясню, что происходит в коде. Во-первых, мы создаем два вида и кладем их один на другой. Эти виды представляют собой два обычных разноцветных квадрата: второй находится на первом. Оба они добавлены к контроллеру вида. Как и в предыдущих примерах, мы добавляем к аниматору поведение тяготения. Это означает, что, как только анимация начнет действовать, виды станут как будто сползать по опорному виду сверху вниз. Мы не задаем границы опорного вида в качестве границ столкновения, а самостоятельно проводим границу столкновения, располагая ее в 100 точках над нижней границей экрана. У нас получается невидимая линия, проходящая по экрану слева направо. Она не позволяет видам бесконечно падать вниз и выходить за пределы опорного вида.

Кроме того, как видите, мы задаем вид с контроллером в качестве делегата поведения столкновения. Таким образом, он получает обновления от этого поведения, сообщающие, произошло ли столкновение и если произошло, то когда. Как только вы узнаете о факте столкновения, то, вероятно, захотите определить, было ли это столкновение с границей (например, созданной нами) или с одним из элементов сцены. Например, если вы создали в опорном виде множество виртуальных стен, а маленькие виды-квадраты могут сталкиваться с этими стенами, то можете реализовать иной эффект (скажем, взрыв), зависящий от того, с чем именно произошло столкновение. О том, с каким элементом произошло столкновение, вы сможете узнать из делегатного метода, вызываемого в контроллере вида и дающего идентификатор той границы, с которой столкнулся элемент. Зная, какой это был объект, мы можем решить, что делать дальше.

В примере мы сравниваем идентификатор, получаемый от поведения столкновения, с константой `kBottomBoundary`, которую присвоили барьеру при создании этого барьера. Создаем для объекта такую анимацию: квадрат под действием тяготения движется по экрану вниз, вплоть до установленной нами границы. Граница гарантирует, что квадрат остановится на расстоянии 100 точек от нижнего края экрана, на проведенной линии.

Одним из самых интересных свойств класса `UIGravityBehavior` является `collisionMode`. Это свойство описывает, как столкновение должно обрабатываться в аниматоре. Например, в предыдущем примере мы рассмотрели типичное столкновение, добавленное в аниматор без изменения значения `collisionMode`. В данном случае это поведение регистрирует столкновения между квадратиками, а также между квадратиками и теми

границами, которые мы провели в опорном виде. Однако мы можем модифицировать это поведение, изменив значение упомянутого свойства. Вот значения, которые можно для него задать:

- `UICollisionBehaviorModeItems` — при таком значении поведение будет регистрировать столкновения между динамическими элементами — в данном случае между движущимися квадратиками;
- `UICollisionBehaviorModeBoundaries` — при этом значении регистрируются столкновения между динамическими элементами и установленными нами границами, расположенными в опорном виде;
- `UICollisionBehaviorModeEverything` — при таком значении регистрируются любые столкновения, независимо от того, участвуют в них сами элементы, элементы и границы или что-либо еще. Это значение для данного свойства задается по умолчанию.

Можно комбинировать рассмотренные ранее значения с помощью побитового оператора ИЛИ и создавать сочетания режимов столкновения, соответствующие поставленным перед вами бизнес-требованиям.

Рекомендую поэкспериментировать со значениями свойства `collisionMode` и в предыдущем примере задать для этого свойства значение `UICollisionBehaviorModeBoundaries`, а потом запустить приложение. Вы увидите, как оба квадратика упадут в нижнюю часть экрана, окажутся на проведенной границе, но не столкнутся, а вдвинутся друг в друга. Дело в том, что код просто проигнорирует столкновение между ними.

## См. также

Раздел 2.1.

# 2.3. Анимирование компонентов пользовательского интерфейса с помощью толчков

## Постановка задачи

Требуется визуально перебрасывать виды с одного места на экране на другое.

## Решение

Инициализируйте объект поведения типа `UIPushBehavior` с помощью относящегося к нему метода `initWithItems:mode:` и передайте ему значение `UIPushBehaviorModeContinuous`. Как только будете готовы толкать элементы под углом, вызовите для

толчка метод `setAngle:`. Этот метод задает угол (в радианах) для данного поведения. Затем потребуется установить *магнитуду*, то есть силу толчка. Эта величина задается с помощью относящегося к толчку поведения `setMagnitude:`. Магнитуда рассчитывается следующим образом: магнитуда величиной 1 точка соответствует ускорению 100 точек/с$^2$, прилагаемому к целевым видам.

## Обсуждение

Толчки, прилагаемые к экранным элементам, очень полезны — особенно толчки, вызывающие непрерывное движение. Допустим, вы разрабатываете приложение-фотоальбом для iPad. В верхней части экрана создали три слайда, каждый из которых соответствует странице альбома, созданной пользователем. В нижней части экрана располагаются различные картинки, которые пользователь может перетаскивать и раскладывать на страницах. Один из способов, позволяющих реализовать для пользователя такую возможность, — добавление к опорному виду регистратора жестов касания (tap gesture recognizer), создание которого рассмотрено в разделе 10.5. Этот регистратор обеспечивает отслеживание пользовательских жестов касания и позволяет перемещать изображения на целевой слайд. Процесс выглядит как перетаскивание. Другой, пожалуй, более оптимальный способ решения этой задачи — использование толчкового поведения, которое разработчики Apple включили в UIKit.

Толчковое поведение относится к типу `UIPushBehavior` и обладает магнитудой и углом. Угол измеряется в радианах, а магнитуда в 1 точку приводит к ускорению движения, равному 100 точек/с$^2$. Толчковые поведения создаются точно так же, как и любые другие: сначала их необходимо инициализировать, а потом добавить к аниматору типа `UIDynamicAnimator`.

В этом примере мы собираемся создать вид и поместить его в центре более крупного вида контроллера. Мы подключим к аниматору поведение столкновений, благодаря чему маленький вид не будет вылетать за пределы большого вида (с контроллером). О том, как работать со столкновениями, мы поговорили в разделе 2.2. Затем добавим регистратор жестов касания (см. раздел 10.5) к контроллеру вида. Этот регистратор будет уведомлять нас о каждом жесте касания, произошедшем на экране.

Когда касание будет зарегистрировано, рассчитаем угол между точкой касания и центром маленького квадратного вида. Так мы получим угол, выраженный в радианах, под которым сможем толкнуть этот квадратный вид. Затем рассчитаем расстояние между точкой касания и центром маленького вида, полученное значение используем в качестве магнитуды толчка. Таким образом, магнитуда будет тем больше, чем дальше от центра квадратного вида находится точка касания.

В данном разделе предполагается, что читатель понимает основы тригонометрии. Но даже если вы с ними не знакомы — ничего страшного, поскольку для работы потребуются лишь те формулы, которые я описываю в примерах кода к этому разделу. На рис. 2.2 показано, как вычисляется угол между двумя точками. Итак, я надеюсь, что объяснение получится достаточно подробным, чтобы вы могли написать собственное решение данной проблемы.

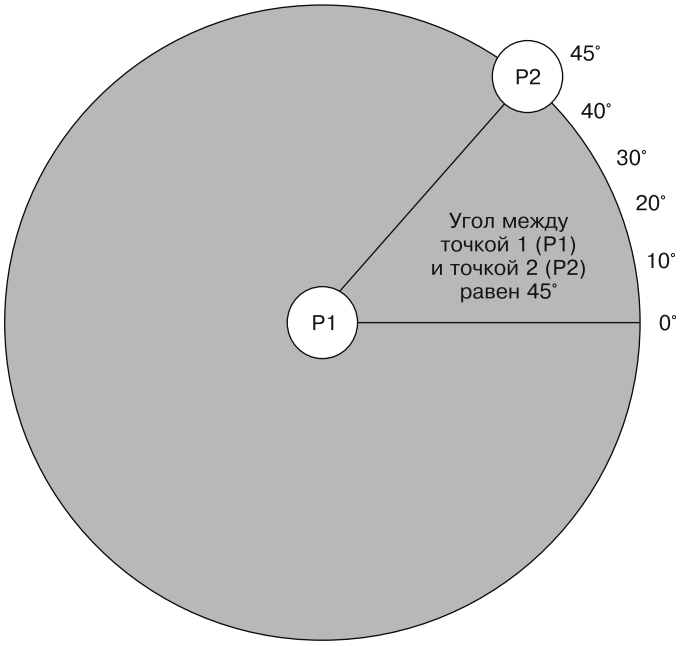


**Рис. 2.2.** Расчет угла между двумя точками

Итак, начнем с определения всех важных свойств нашего контроллера вида:

```
#import "ViewController.h"

@interface ViewController ()
@property (nonatomic, strong) UIView *squareView;
@property (nonatomic, strong) UIDynamicAnimator *animator;
@property (nonatomic, strong) UIPushBehavior *pushBehavior;
@end
@implementation ViewController

<# Остальной ваш код находится здесь #>
```

В этом примере мы добавим к аниматору поведение столкновения и толчковое поведение. Толчковое поведение добавляется к классу в качестве свойства, а поведение столкновений — просто как локальная переменная. Дело в том, что, как только мы добавим к аниматору поведение столкновения, именно аниматор будет вычислять все столкновения с границами опорного вида и нам больше не придется ссылаться на это поведение столкновений. Однако если говорить о толчковом поведении, то при обработке касаний придется обновлять это толчковое поведение, чтобы графический элемент подталкивался к точке касания. Вот почему нам требуется связь касания с толчковым поведением, но не требуется такая связь со столкновениями.

Далее напишем метод, создающий маленький квадратный вид и помещающий его в центре большого вида с контроллером:

```
- (void) createSmallSquareView{
  self.squareView =
  [[UIView alloc] initWithFrame:
   CGRectMake(0.0f, 0.0f, 80.0f, 80.0f)];
  self.squareView.backgroundColor = [UIColor greenColor];

  self.squareView.center = self.view.center;

  [self.view addSubview:self.squareView];
}
```

Затем применим регистратор жестов касания, чтобы обнаруживать прикосновения к виду с контроллером:

```
- (void) createGestureRecognizer{
  UITapGestureRecognizer *tapGestureRecognizer =
  [[UITapGestureRecognizer alloc] initWithTarget:self
                                          action:@selector(handleTap:)];
  [self.view addGestureRecognizer:tapGestureRecognizer];
}
```

Эти методы выполняют за нас всю необходимую работу. Позже, когда вид отобразится на экране, мы будем вызывать эти методы и они будут действовать.

И не забудем написать метод, который будет задавать поведение столкновения и толчковое поведение:

```
- (void) createAnimatorAndBehaviors{
  self.animator = [[UIDynamicAnimator alloc]
                   initWithReferenceView:self.view];

  /* Создаем обнаружение столкновений */
  UICollisionBehavior *collision = [[UICollisionBehavior alloc]
                                    initWithItems:@[self.squareView]];
  collision.translatesReferenceBoundsIntoBoundary = YES;

  self.pushBehavior = [[UIPushBehavior alloc]
                       initWithItems:@[self.squareView]
                       mode:UIPushBehaviorModeContinuous];

  [self.animator addBehavior:collision];
  [self.animator addBehavior:self.pushBehavior];
}
```

Подробнее о поведении столкновений рассказано в разделе 2.2. Как только мы запрограммируем все эти методы, нам понадобится вызывать их, когда вид появится на экране:

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

  [self createGestureRecognizer];
```

```
  [self createSmallSquareView];
  [self createAnimatorAndBehaviors];

}
```

Отлично. Теперь, взглянув на файл реализации метода `createGestureRecognizer`, вы увидите, что мы устанавливаем регистратор жестов касаний в методе контроллера вида — этот метод называется `handleTap:`. В методе `handleTap:` вычисляем расстояние между центральной точкой маленького квадратного вида и той точкой опорного вида, до которой дотронулся пользователь. В результате имеем магнитуду силы толчка. Кроме того, рассчитаем угол между центром маленького квадратного вида и точкой касания, чтобы определить угол толчка:

```
- (void) handleTap:(UITapGestureRecognizer *)paramTap{

  /* Получаем угол между центральной точкой квадратного вида
   и точкой касания */

  CGPoint tapPoint = [paramTap locationInView:self.view];
  CGPoint squareViewCenterPoint = self.squareView.center;

  /* Вычисляем угол между центральной точкой квадратного вида
   и точкой касания, чтобы определить угол толчка

  Формула для определения угла между двумя точками:
  arc tangent 2((p1.x - p2.x), (p1.y - p2.y)) */

  CGFloat deltaX = tapPoint.x - squareViewCenterPoint.x;
  CGFloat deltaY = tapPoint.y - squareViewCenterPoint.y;
  CGFloat angle = atan2(deltaY, deltaX);
  [self.pushBehavior setAngle:angle];

  /* Используем расстояние между точкой касания и центром квадратного
   вида для вычисления магнитуды толчка

   Формула определения расстояния:
   Квадратный корень из ((p1.x - p2.x)^2 + (p1.y - p2.y)^2) */

  CGFloat distanceBetweenPoints =
  sqrt(pow(tapPoint.x - squareViewCenterPoint.x, 2.0) +
       pow(tapPoint.y - squareViewCenterPoint.y, 2.0));
  [self.pushBehavior setMagnitude:distanceBetweenPoints / 200.0f];
}
```

Не буду чрезмерно углубляться в тригонометрию, но в этом коде используется простая формула, изучаемая в школьном курсе. По этой формуле рассчитывается угол в радианах между двумя точками. Также применяется теорема Пифагора, позволяющая узнать расстояние между двумя точками. Эти формулы вы найдете, взглянув на комментарии, которые я оставил в коде. Если же хотите подробнее разобраться с такими понятиями, как углы и радианы, рекомендую проштудировать учебник по тригонометрии.

Теперь, запустив приложение, вы сначала увидите маленький зеленый квадрат в центре экрана. Дотроньтесь до экрана в любой точке поля, окружающего квадрат (белое пространство), чтобы зеленый квадрат (вид) стал двигаться. В данном примере я беру расстояние между точкой касания и центром квадрата и делю его на 200, чтобы получить реалистичную магнитуду толчка, но вы в данном примере можете увеличить ускорение, выбрав, скажем, значение 100, а не 200. Всегда лучше экспериментировать с разными числовыми значениями, чтобы подобрать оптимальный вариант для *вашего* приложения.

## См. также

Раздел 2.2.

# 2.4. Прикрепление нескольких динамических элементов друг к другу

## Постановка задачи

Требуется прикреплять друг к другу динамические элементы, например виды, так, чтобы движения одного вида автоматически приводили в движение второй. В качестве альтернативы можно прикреплять динамический элемент к точке привязки, чтобы при движении этой точки (в результате действий приложения или пользователя) этот элемент автоматически перемещался вместе с ней.

## Решение

Инстанцируйте поведение прикрепления, относящееся к типу `UIAttachmentBehavior`, с помощью метода экземпляра `initWithItem:point:attachedToAnchor:` этого класса. Добавьте это поведение к аниматору (см. раздел 2.0), отвечающему за динамику и физику движения.

## Обсуждение

На первый взгляд поведение прикрепления может показаться непонятным. Оно сводится к следующему: вы можете задать на экране точку привязки, а затем заставить точку следовать за этой привязкой. Но я хотел бы обсудить эту возможность подробнее.

Допустим, у вас на столе лежит большая фотография. Если вы поставите указательный палец в верхний правый угол фотографии и начнете совершать им вращательные движения, то фотография, возможно, также будет вертеться на столе вместе с вашим пальцем. Такое же реалистичное поведение вы можете создать и в iOS, воспользовавшись поведением прикрепления из UIKit.

В этом примере мы собираемся создать такой эффект, который продемонстрирован на рис. 2.3.

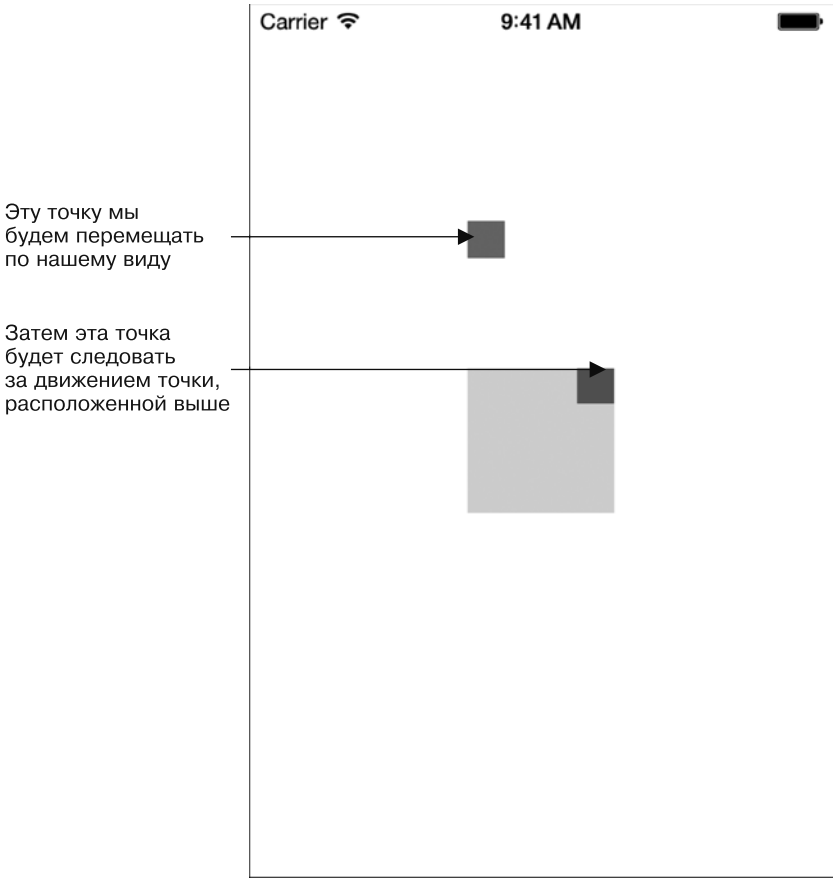


**Рис. 2.3.** Именно такого эффекта мы хотим добиться в данном разделе с помощью поведения прикрепления

Как видите, на экране находятся три вида. Основной вид расположен в центре, в правом верхнем углу этого вида есть еще один вид, более мелкий. Маленький вид — это и есть тот элемент, который будет следовать за точкой привязки, по принципу, который я описал в примере с фотографией. Наконец, необходимо отметить, что точка привязки в данном примере будет перемещаться по экрану под действием жеста панорамирования и регистратора соответствующих жестов (см. раздел 10.3). Затем в результате таких движений станет двигаться большой вид, расположенный в центре экрана. Итак, начнем с определения необходимых свойств контроллера вида:

```
#import "ViewController.h"

@interface ViewController ()
@property (nonatomic, strong) UIView *squareView;
@property (nonatomic, strong) UIView *squareViewAnchorView;
@property (nonatomic, strong) UIView *anchorView;
```

```
@property (nonatomic, strong) UIDynamicAnimator *animator;
@property (nonatomic, strong) UIAttachmentBehavior *attachmentBehavior;
@end

@implementation ViewController

<# Оставшаяся часть кода контроллера вида находится здесь #>
```

Далее нам потребуется создать маленький квадратный вид. Но на этот раз мы поместим внутрь него еще один вид. Маленький вид, который будет располагаться в правом верхнем углу родительского вида, мы фактически соединим с точкой привязки поведения прикрепления, как было показано в примере с фотографией:

```
- (void) createSmallSquareView{
  self.squareView =
  [[UIView alloc] initWithFrame:
   CGRectMake(0.0f, 0.0f, 80.0f, 80.0f)];

  self.squareView.backgroundColor = [UIColor greenColor];
  self.squareView.center = self.view.center;

  self.squareViewAnchorView = [[UIView alloc] initWithFrame:
                               CGRectMake(60.0f, 0.0f, 20.0f, 20.0f)];
  self.squareViewAnchorView.backgroundColor = [UIColor brownColor];
  [self.squareView addSubview:self.squareViewAnchorView];

  [self.view addSubview:self.squareView];
}
```

Далее создадим вид с точкой привязки:

```
- (void) createAnchorView{

  self.anchorView = [[UIView alloc] initWithFrame:
                     CGRectMake(120.0f, 120.0f, 20.0f, 20.0f)];
  self.anchorView.backgroundColor = [UIColor redColor];
  [self.view addSubview:self.anchorView];

}
```

После этого потребуется создать регистратор жестов панорамирования и аниматор, как мы уже делали в предыдущих разделах этой главы:

```
- (void) createGestureRecognizer{
  UIPanGestureRecognizer *panGestureRecognizer =
  [[UIPanGestureRecognizer alloc] initWithTarget:self
                                          action:@selector(handlePan:)];
  [self.view addGestureRecognizer:panGestureRecognizer];
}

- (void) createAnimatorAndBehaviors{

  self.animator = [[UIDynamicAnimator alloc]
```

```
                    initWithReferenceView:self.view];

  /* Создаем распознавание столкновений */
  UICollisionBehavior *collision = [[UICollisionBehavior alloc]
                                    initWithItems:@[self.squareView]];
  collision.translatesReferenceBoundsIntoBoundary = YES;

  self.attachmentBehavior = [[UIAttachmentBehavior alloc]
                             initWithItem:self.squareView
                             point:self.squareViewAnchorView.center
                             attachedToAnchor:self.anchorView.center];
  [self.animator addBehavior:collision];
  [self.animator addBehavior:self.attachmentBehavior];
}

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

  [self createGestureRecognizer];
  [self createSmallSquareView];
  [self createAnchorView];
  [self createAnimatorAndBehaviors];
}
```

Как видите, мы реализуем поведение привязки с помощью его метода экземпляра `initWithItem:point:attachedToAnchor:`. Этот метод принимает следующие параметры:

- `initWithItem` — динамический элемент (в нашем примере — вид), который должен быть подключен к точке привязки;
- `point` — точка внутри динамического элемента, которая должна быть соединена с точкой привязки. В данном поведении центральная точка элемента используется для установки соединения с точкой привязки. Но вы можете изменить этот параметр, присвоив ему другое значение;
- `attachedToAnchor` — сама точка привязки, измеряемая как значение `CGPoint`.

Теперь, когда мы соединили верхний правый угол квадратного вида с точкой привязки (представленной как вид точки привязки), необходимо продемонстрировать, что, двигая точку привязки, мы опосредованно будем двигать и квадратный вид. Вернемся к методу `createGestureRecognizer`, написанному ранее. Там мы задействовали регистратор жестов касания, который будет отслеживать движение пальца пользователя по экрану. Мы решили обрабатывать регистратор жестов в методе `handlePan:` вида и реализуем этот метод так:

```
(void) handlePan:(UIPanGestureRecognizer *)paramPan{

  CGPoint tapPoint = [paramPan locationInView:self.view];
  [self.attachmentBehavior setAnchorPoint:tapPoint];
  self.anchorView.center = tapPoint;

}
```

Здесь мы обнаруживаем в нашем виде движущуюся точку, а потом перемещаем в нее точку привязки. После того как мы это сделаем, произойдет прикрепление и мы сможем двигать также маленький квадрат.

## См. также

Разделы 2.0 и 10.3.

# 2.5. Добавление эффекта динамического зацепления к компонентам пользовательского интерфейса

## Постановка задачи

С помощью анимации вы хотите прикрепить определенный вид, находящийся в вашем пользовательском интерфейсе, к конкретному месту на экране. При этом должна проявляться эластичность, напоминающая реальный эффект защелкивания. Таким образом, когда элемент пользовательского интерфейса прикрепляется к определенной точке экрана, пользователь ощущает, что этот элемент обладает встроенной эластичностью.

## Решение

Инстанцируйте объект типа `UISnapBehavior` и добавьте его к аниматору типа `UIDynamicAnimator`.

## Обсуждение

Чтобы по-настоящему понять, как работает динамика зацепления, представим себе небольшое количество желе, смазанное маслом и лежащее на очень гладком столе. К желе прикреплена струна. Представляю, насколько странным вам это кажется. Но следите за мыслью. Допустим, я стою возле стола и тяну за струну, чтобы желе переместилось из исходной точки на столе в другую, выбранную вами. Поскольку желе со всех сторон покрыто маслом, оно будет плавно двигаться в этом направлении. Но раз это желе, оно, оказавшись в выбранной вами точке, еще некоторое время будет колыхаться. Именно такое поведение реализуется с помощью класса `UISnapBehavior`.

Один из способов практического применения такого эффекта заключается в следующем: если у вас есть приложение, на экране с которым расположено несколько видов, то, возможно, вы захотите предоставить пользователю возможность передвигать эти виды по экрану по своему желанию и самостоятельно настраивать компоновку интерфейса. Эту задачу вполне можно решить с помощью приемов, описанных в разделе 2.3, но такой вариант получится слишком негибким. Вообще техники из раздела 2.3 предназначены для решения иных задач. В этом разделе у нас есть экран, и мы добиваемся того, чтобы пользователь мог прикоснуться к любому

виду на экране и переместить его. Но потом мы зацепим этот вид, ассоциировав его с точкой, в которой произошло касание.

В данном рецепте мы собираемся создать маленький вид в центре основного вида контроллера, а потом прикрепить регистратор жестов касания (см. раздел 10.5) к виду с контроллером. Всякий раз, когда пользователь прикасается к экрану в какой-то точке, мы будем зацеплять за эту точку маленький квадратный вид. Итак, приступим к определению необходимых свойств вида с контроллером:

```
#import "ViewController.h"

@interface ViewController ()
@property (nonatomic, strong) UIView *squareView;
@property (nonatomic, strong) UIDynamicAnimator *animator;
@property (nonatomic, strong) UISnapBehavior *snapBehavior;
@end
@implementation ViewController

<# Остальной ваш код находится здесь #>
```

Далее напишем метод, который будет создавать регистратор жестов касания:

```
- (void) createGestureRecognizer{

  UITapGestureRecognizer *tap = [[UITapGestureRecognizer alloc]
                                 initWithTarget:self
                                 action:@selector(handleTap:)];
  [self.view addGestureRecognizer:tap];

}
```

Как и в предыдущих разделах, нам также понадобится создать маленький вид в центре экрана. Я выбрал для этой цели именно центр, но вы можете использовать в таком качестве другую точку. Этот вид мы будем сцеплять с теми точками экрана, к которым прикоснется пользователь. Итак, вот метод для создания этого вида:

```
- (void) createSmallSquareView{
  self.squareView =
  [[UIView alloc] initWithFrame:
   CGRectMake(0.0f, 0.0f, 80.0f, 80.0f)];

  self.squareView.backgroundColor = [UIColor greenColor];
  self.squareView.center = self.view.center;

  [self.view addSubview:self.squareView];
}
```

Переходим к созданию аниматора (см. раздел 2.0), после чего прикрепляем к нему поведение зацепления. Инициализируем поведение зацепления типа `UISnapBehavior` с помощью метода `initWithItem:snapToPoint:`. Этот метод принимает два параметра:

- `initWithItem` — динамический элемент (в данном случае наш вид), к которому должно применяться поведение зацепления. Как и другие динамические по-

ведения пользовательского интерфейса, этот элемент должен соответствовать протоколу UIDynamicItem. Все экземпляры UIView по умолчанию соответствуют этому протоколу, поэтому все нормально;

- snapToPoint — точка опорного вида (см. раздел 2.0), за которую должен зацепляться динамический элемент.

Следует сделать одно важное замечание о таком зацеплении: чтобы оно работало с конкретным элементом, к аниматору уже должен быть добавлен как минимум один экземпляр зацепления для этого элемента — кроме того экземпляра, который удерживает элемент на текущей позиции. После этого все последующие зацепления будут работать правильно. Позвольте это продемонстрировать. Сейчас мы реализуем метод, который будет создавать поведение зацепления и аниматор, а потом добавлять это поведение к аниматору:

```
- (void) createAnimatorAndBehaviors{
  self.animator = [[UIDynamicAnimator alloc]
                   initWithReferenceView:self.view];

  /* Создаем обнаружение столкновений */
  UICollisionBehavior *collision = [[UICollisionBehavior alloc]
                                    initWithItems:@[self.squareView]];
  collision.translatesReferenceBoundsIntoBoundary = YES;

  [self.animator addBehavior:collision];

  /* Пока зацепляем квадратный вид с его актуальным центром */
  self.snapBehavior = [[UISnapBehavior alloc]
                       initWithItem:self.squareView
  snapToPoint:self.squareView.center];
  self.snapBehavior.damping = 0.5f; /* Medium oscillation */
  [self.animator addBehavior:self.snapBehavior];
}
```

Как видите, здесь мы зацепляем небольшой квадратный вид, связывая его с текущим центром, — в сущности, просто оставляем его на месте. Позже, когда мы регистрируем на экране жесты касания, мы обновляем поведение зацепления. Кроме того, необходимо отметить, что мы задаем для этого поведения свойство damping. Это свойство будет управлять эластичностью, с которой элемент будет зацеплен за точку. Чем выше значение, тем меньше эластичность, соответственно, тем слабее «колышется» элемент. Здесь можно задать любое значение в диапазоне от 0 до 1. Теперь, когда вид появится на экране, вызовем эти методы, чтобы инстанцировать маленький квадратный вид, установить регистратор жестов касания, а также настроить аниматор и поведение зацепления:

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

  [self createGestureRecognizer];
  [self createSmallSquareView];
  [self createAnimatorAndBehaviors];
}
```

После создания регистратора жестов касания в методе `createGestureRecognizer` вида с контроллером мы приказываем регистратору сообщать о таких касаниях методу `handleTap:` вида с контроллером. В этом методе мы получаем точку, в которой пользователь прикоснулся к экрану, после чего обновляем поведение зацепления.

Здесь необходимо отметить, что вы не сможете просто обновить существующее поведение — потребуется повторно его инстанцировать. Итак, прежде, чем мы инстанцируем новый экземпляр поведения зацепления, понадобится удалить старый экземпляр (при его наличии), а потом добавить к аниматору новый. У каждого аниматора может быть всего одно поведение зацепления, ассоциированное с конкретным динамическим элементом, в данном случае с маленьким квадратным видом. Если добавить к одному и тому же аниматору несколько поведений зацепления, относящихся к одному и тому же динамическому элементу, то аниматор проигнорирует все эти поведения, так как не будет знать, какое из них выполнять первым. Поэтому, чтобы поведения зацепления работали, сначала удалите все зацепления для этого элемента из вашего аниматора, воспользовавшись его методом `removeBehavior:`, а потом добавьте новое поведение зацепления следующим образом:

```
- (void) handleTap:(UITapGestureRecognizer *)paramTap{

  /* Получаем угол между центром квадратного вида и точкой касания */

  CGPoint tapPoint = [paramTap locationInView:self.view];

  if (self.snapBehavior != nil){
    [self.animator removeBehavior:self.snapBehavior];
  }

  self.snapBehavior = [[UISnapBehavior alloc] initWithItem:self.squareView
                       snapToPoint:tapPoint];
  self.snapBehavior.damping = 0.5f; /* Средняя осцилляция */
  [self.animator addBehavior:self.snapBehavior];
}
```

## См. также

Разделы 2.0 и 10.5.

# 2.6. Присваивание характеристик динамическим эффектам

## Постановка задачи

Вероятно, вас устраивает стандартная физика, по умолчанию встроенная в динамические поведения из UIKit. Но при работе с элементами, которыми вы управляете с помощью динамических поведений, может потребоваться присваивать этим элементам и иные характеристики, например массу и эластичность.

## Решение

Инстанцируйте объект типа `UIDynamicItemBehavior` и присвойте ему ваши динамические элементы. После инстанцирования пользуйтесь различными свойствами этого класса для изменения характеристик динамических элементов. Затем добавьте это поведение к аниматору (см. раздел 2.0) — и все остальное аниматор сделает за вас.

## Обсуждение

Динамические поведения отлично подходят для добавления реалистичной физики к элементам, соответствующим протоколу `UIDynamicItem`, например ко всем видам типа `UIView`. Но в некоторых приложениях вам может понадобиться явно указать характеристики конкретного элемента. Например, в приложении, где используются эффекты тяготения и столкновения (см. разделы 2.1 и 2.2), вы, возможно, захотите указать, что один из элементов на экране, подвергающийся действию тяготения и столкновениям, должен отскакивать от границы сильнее, чем другой элемент. В другом случае, возможно, захотите указать, что в ходе воплощения различных визуальных эффектов, применяемых аниматором к элементу, этот элемент вообще не должен вращаться.

Подобные задачи решаются без труда с помощью экземпляров класса `UIDynamicItemBehavior`. Эти экземпляры также являются динамическими поведениями, и мы можем добавлять их к аниматору с помощью метода экземпляра `addBehavior:`, относящегося к классу `UIDynamicAnimator`, — в этой главе мы так уже делали. Когда вы инициализируете экземпляр этого класса, вы вызываете метод-инициализатор `initWithItems:` и передаете ваш вид либо какой угодно объект, соответствующий протоколу `UIDynamicItem`. В качестве альтернативы можете инициализировать экземпляр элемента с динамическим поведением с помощью метода `init`, а потом добавлять к этому поведению разнообразные объекты, пользуясь методом `addItem:`.

Экземпляры класса `UIDynamicItemBehavior` обладают настраиваемыми свойствами, которые вы можете корректировать, чтобы модифицировать поведение динамических элементов (например, видов). Далее перечислены и объяснены некоторые наиболее важные свойства этого класса.

- `allowsRotation` — логическое значение. Если оно равно `YES`, то, как понятно из названия, это свойство позволяет аниматору вращать динамические элементы в ходе применения к ним визуальных эффектов. В идеале вы должны устанавливать значение этого свойства в `YES`, если желаете имитировать реалистичную физику, но если по каким-то причинам в приложении необходимо гарантировать, что определенный элемент ни при каких условиях вращаться не будет, задайте для этого свойства значение `NO` и прикрепите элемент к этому поведению.
- `resistance` — сопротивление элемента движению. Это свойство может иметь значения в диапазоне от `0` до `CGFLOAT_MAX`. Чем выше значение, тем сильнее будет сопротивление, оказываемое элементом воздействующим на него силам (тем силам, которые вы к нему прикладываете). Например, если вы добавите к аниматору поведение тяготения и создадите в центре экрана вид с сопротивлением `CGFLOAT_MAX`, то тяготение не сможет сдвинуть этот вид к центру экрана. Вид просто останется там, где вы его создали.

- `friction` — это значение с плавающей точкой в диапазоне от `0` до `1`. Оно указывает силу трения, которая должна действовать на края данного элемента, когда другие элементы соударяются с ним или проскальзывают по его краю. Чем выше значение, тем больше сила трения, применяемая к элементу.

  Чем больше будет сила трения, которую вы применяете к элементу, тем более *клейким* он становится. Такая склонность элемента к прилипанию выражается в том, что когда другие элементы с ним сталкиваются, они задерживаются около клейкого элемента чуть дольше, чем обычно. Представьте себе, как действует трение на шины автомобиля. Чем больше сила трения между шинами и асфальтом, тем медленнее будет двигаться автомобиль, но тем лучше будет сцепление шин с дорогой, даже с довольно скользкой. Именно такой тип трения это свойство позволяет присваивать вашим элементам.
- `elasticity` — это значение с плавающей точкой в диапазоне от `0` до `1`. Оно указывает, насколько эластичным должен быть элемент. Чем выше это значение, тем более пластичным и «желеобразным» будет этот элемент с точки зрения аниматора. О том, что такое эластичность, подробно рассказано в разделе 2.5.
- `density` — это значение с плавающей точкой в диапазоне от `0` до `1` (по умолчанию применяется значение `1`). Оно не используется непосредственно для воздействия на поведение динамических поведений элементов, но с его помощью аниматор рассчитывает массу объектов и то, как эта масса отражается на визуальных эффектах. Например, если вы столкнете один элемент с другим (см. раздел 2.3), но у одного из них будет плотность 1, а у другого — 0,5, то при одинаковых высоте и ширине обоих элементов масса первого элемента будет больше, чем масса второго. Аниматор вычисляет массу элементов, исходя из их плотности и размеров экрана. Поэтому если вы столкнете маленький вид с высокой плотностью с большим видом с очень низкой плотностью, то маленький вид, в зависимости от конкретного размера и плотности этого элемента, может быть воспринят аниматором как более массивный объект, нежели крупный вид. В таком случае, аниматор может сильнее оттолкнуть тот элемент, который на экране кажется более крупным, тогда как толчок, сообщаемый крупным элементом мелкому, получится не столь значительным.

Перейдем к примеру. Он отчасти основан на примере, рассмотренном в разделе 2.2. В примере из этого раздела мы собираемся расположить один вид на другом, но тот вид, который находится снизу, будет очень эластичным, а эластичность вида, расположенного сверху, будет сравнительно невысока. Таким образом, когда оба вида упадут на дно экрана, где столкнутся с нижней границей, нижний вид отскочит от нее значительно сильнее, чем верхний. Итак, начнем с определения аниматора и других свойств контроллера вида:

```
#import "ViewController.h"

@interface ViewController ()
@property (nonatomic, strong) UIDynamicAnimator *animator;
@end

@implementation ViewController

<# Оставшаяся часть вашего кода находится здесь #>
```

Далее напишем удобный метод, с помощью которого сможем создавать виды с заранее заданными центральной точкой и цветом фона. Этот метод мы используем для создания двух очень похожих видов с разными центральными точками и окрашенных в разные фоновые цвета:

```
- (UIView *) newViewWithCenter:(CGPoint)paramCenter
               backgroundColor:(UIColor *)paramBackgroundColor{

  UIView *newView =
  [[UIView alloc] initWithFrame:
   CGRectMake(0.0f, 0.0f, 50.0f, 50.0f)];
  newView.backgroundColor = paramBackgroundColor;
  newView.center = paramCenter;

  return newView;

}
```

Теперь, как только основной вид отобразится на экране, создадим два этих вида и также выведем их на дисплей:

```
UIView *topView = [self newViewWithCenter:CGPointMake(100.0f, 0.0f)
                          backgroundColor:[UIColor greenColor]];
UIView *bottomView = [self newViewWithCenter:CGPointMake(100.0f, 50.0f)
                            backgroundColor:[UIColor redColor]];

[self.view addSubview:topView];
[self.view addSubview:bottomView];
```

Далее добавим к видам поведение тяготения — этому мы научились в разделе 2.1:

```
self.animator = [[UIDynamicAnimator alloc]
                 initWithReferenceView:self.view];

/* Создаем тяготение */
UIGravityBehavior *gravity = [[UIGravityBehavior alloc]
                              initWithItems:@[topView, bottomView]];
[self.animator addBehavior:gravity];
```

Мы не хотим, чтобы виды выпадали за пределы экрана, достигнув его дна. Поэтому воспользуемся знаниями, приобретенными в разделе 2.2, и зададим для аниматора нижнюю границу, а также запрограммируем поведение столкновения:

```
/* Создаем обнаружение столкновений */
UICollisionBehavior *collision = [[UICollisionBehavior alloc]
                                  initWithItems:@[topView, bottomView]];
collision.translatesReferenceBoundsIntoBoundary = YES;

[self.animator addBehavior:collision];
```

Наконец, очень важно добавить видам динамическое поведение, чтобы сделать верхний вид менее эластичным, чем нижний:

```
/* Теперь указываем эластичность элементов */
UIDynamicItemBehavior *moreElasticItem = [[UIDynamicItemBehavior alloc]
                                          initWithItems:@[bottomView]];

moreElasticItem.elasticity = 1.0f;
UIDynamicItemBehavior *lessElasticItem = [[UIDynamicItemBehavior alloc]
                                          initWithItems:@[topView]];
lessElasticItem.elasticity = 0.5f;
[self.animator addBehavior:moreElasticItem];
[self.animator addBehavior:lessElasticItem];
```

Итак, можете запустить приложение и посмотреть, как виды будут отскакивать от нижней границы экрана, как только ударятся об нее (рис. 2.4).

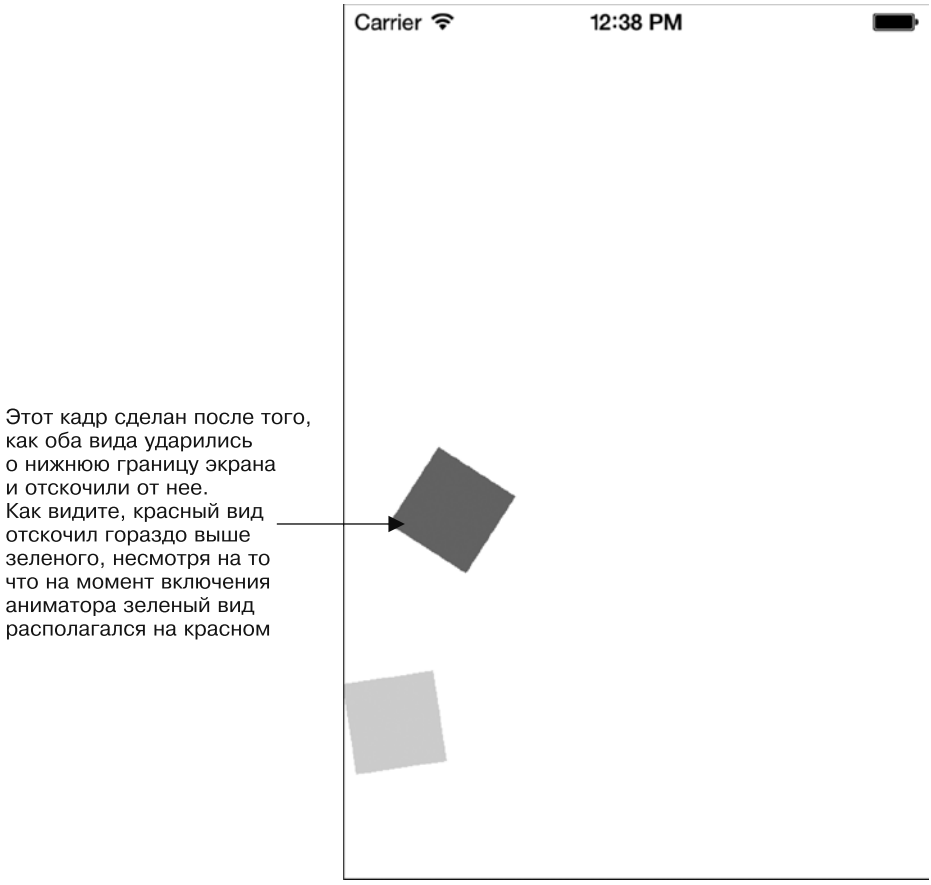


**Рис. 2.4.** Один вид эластичнее другого

## См. также

Раздел 2.0.

# 3 Автоматическая компоновка и язык визуального форматирования

## 3.0. Введение

Выравнивание компонентов пользовательского интерфейса всегда было для программиста большой проблемой. В большинстве контроллеров видов в сложных приложениях для iOS содержится множество кода, решающего такие якобы тривиальные задачи, как упорядочение на экране фрейма с графическими элементами, выравнивание компонентов по горизонтали и вертикали и обеспечение того, что компоненты будут нормально выглядеть в различных версиях iOS. Причем проблема не только в этом, ведь многие программисты желают пользоваться одними и теми же контроллерами видов на разных устройствах, например на iPhone и iPad. Из-за этого код дополнительно усложняется. Apple упростила для нас решение таких задач, предоставив возможность автоматической компоновки (Auto Layout). Автоматическая компоновка, давно применявшаяся в OS X, теперь реализована и в iOS. Чуть позже мы подробно поговорим об автоматической компоновке, но для начала я позволю себе краткое введение и расскажу, для чего она нужна.

Допустим, у вас есть кнопка, которая обязательно должна находиться в центре экрана. Отношение между центром кнопки и центром вида, в котором она находится, можно упрощенно описать следующим образом:

- свойство кнопки `center.x` равно свойству вида `center.x`;
- свойство кнопки `center.y` равно свойству вида `center.y`.

Разработчики Apple заметили, что многие проблемы, связанные с позиционированием элементов пользовательского интерфейса, решаемы с помощью простой формулы:

```
object1.property1 = (object2.property2 * multiplier) + constant value
```

Например, воспользовавшись этой формулой, я могу без труда центрировать кнопку в ее вышестоящем виде, вот так:

```
button.center.x = (button.superview.center.x * 1) + 0
button.center.y = (button.superview.center.y * 1) + 0
```

С помощью этой же формулы вы можете делать некоторые по-настоящему отличные вещи при разработке пользовательского интерфейса приложений для iOS — вещи, которые ранее были просто неосуществимы. В iOS SDK вышеупомянутая формула обернута в класс, который называется `NSLayoutConstraint`. Каждый экземпляр этого класса соответствует ровно одному ограничению. Например, если вы хотите расположить кнопку в центре вида, владеющего этой кнопкой, то требуется центрировать координаты *x* и *y* этой кнопки. Таким образом, речь идет о создании двух ограничений. Но далее в этой главе мы познакомимся с языком визуального форматирования (Visual Format Language). Он отлично дополняет язык программирования для iOS и еще сильнее упрощает работу с макетами пользовательского интерфейса.

Ограничения можно создавать с помощью так называемых перекрестных видов. Например, если в одном виде у вас находится две кнопки и вы хотите, чтобы по вертикали между ними было 100 точек свободного пространства, то нужно создать ограничение, благодаря которому выполнялось бы это правило, но добавить его к общему родителю обеих этих кнопок. Скорее всего, это владеющий ими вид. Вот эти правила.

- Если ограничение находится между двумя видами, которые располагаются в общем родительском виде (то есть у обоих этих видов один и тот же вышестоящий родительский вид), добавьте ограничения к родительскому виду.
- Если ограничение находится между видом и его родительским видом, добавьте ограничение к родительскому виду.
- Если ограничение находится между двумя видами, которые не располагаются в общем родительском виде, добавьте это ограничение к общему предку интересующих вас видов.

На рис. 3.1 показано, как именно действуют эти ограничения.

Ограничения создаются с помощью метода класса `constraintWithItem:attribute:relatedBy:toItem:attribute:multiplier:constant:`, который относится к классу `NSLayoutConstraint`. Этот метод принимает следующие параметры:

- `constraintWithItem` — параметр типа `id`. Он соответствует объекту `object1` в формуле, рассмотренной ранее;
- `attribute` — этот параметр представляет свойство `property1` в вышеупомянутой формуле и должен относиться к типу `NSLayoutAttribute`;
- `relatedBy` — параметр соответствует знаку равенства в нашей формуле. Значение этого параметра относится к типу `NSLayoutRelation` и, как вы вскоре убедитесь, может выступать не только в качестве знака равенства, но и в роли знаков «больше» и «меньше». Мы подробно обсудим эти нюансы в данной главе;
- `toItem` — это параметр типа `id`. Он соответствует объекту `object2` в формуле, рассмотренной ранее;
- `attribute` — параметр представляет свойство `property2` в вышеупомянутой формуле и должен относиться к типу `NSLayoutAttribute`;

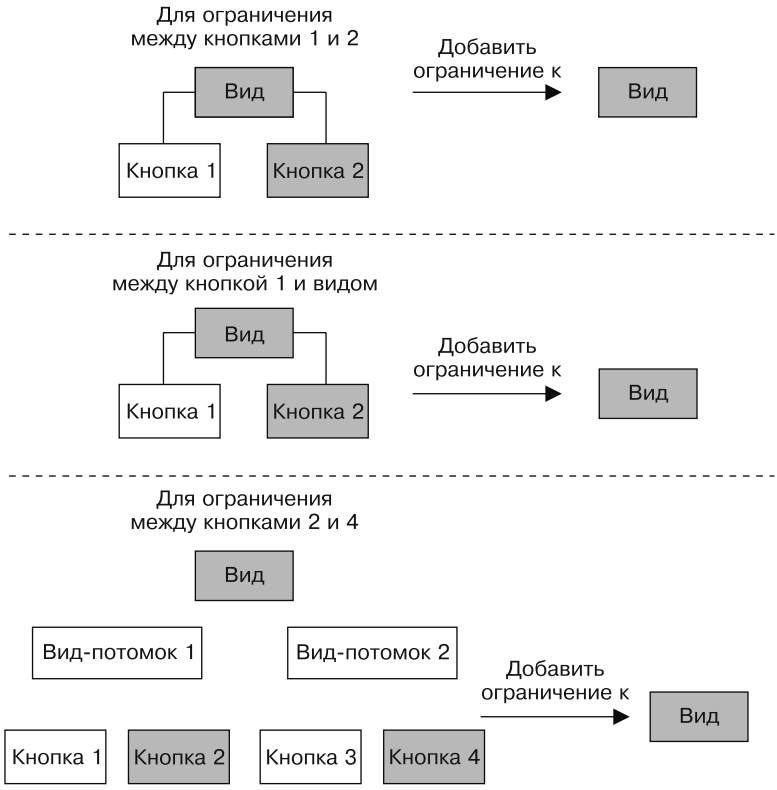


**Рис. 3.1.** Отношения между ограничениями и видами, к которым эти ограничения должны добавляться

- `multiplier` — это параметр типа `CGFloat`, представляющий множитель в нашей формуле;
- `constant` — параметр также относится к типу `CGFloat` и представляет константу в формуле.

После создания ограничений вы сможете просто добавить их к соответствующему виду (рис. 3.1), воспользовавшись одним из следующих методов класса `UIView`:

- `addConstraint:` — метод позволяет добавить к виду одно ограничение типа `NSLayoutConstraint`;
- `addConstraints:` — этот метод позволяет добавить к виду массив ограничений. Ограничения должны относиться к типу `NSLayoutConstraint`, но в данном случае они будут обернуты в массив типа `NSArray`.

Автоматическая компоновка позволяет решать разнообразные задачи, в чем вы убедитесь в оставшейся части этой главы. Тем не менее чем подробнее вы будете знакомиться с этой темой, тем очевиднее будет становиться следующий факт: применяя автоматическую компоновку, вы вынуждены создавать все новые ограничения типа `NSLayoutConstraint`. Из-за этого ваш код будет разрастаться, а поддержка его — постоянно усложняться. Именно поэтому компания Apple разработала язык

визуального форматирования, на котором можно описывать ограничения, пользуясь обычными символами ASCII. Например, если у вас есть две кнопки и вы хотите, чтобы по горизонтали эти кнопки всегда отстояли друг от друга на 100 точек, то нужно написать на языке визуального форматирования подобный код:

```
[button1]-100-[button2]
```

Ограничения, выражаемые на языке визуального форматирования, создаются с помощью метода класса `constraintsWithVisualFormat:options:metrics:views:`, относящегося к классу `NSLayoutConstraint`. Вот краткое описание каждого из параметров этого метода:

- `constraintsWithVisualFormat` — выражение на языке визуального форматирования, записанное как `NSString`;
- `options` — параметр типа `NSLayoutFormatOptions`. При работе с языком визуального форматирования этому параметру обычно передается значение 0;
- `metrics` — словарь констант, которые вы используете в выражении на языке визуального форматирования. Пока ради упрощения примеров будем передавать этому параметру значение `nil`;
- `views` — это словарь видов, для которых вы написали ограничение в первом параметре данного метода. Чтобы создать такой словарь, просто воспользуйтесь функцией `NSDictionaryOfVariableBindings` из языка C и передайте этому методу ваши новые объекты. Ключи в этом словаре — это названия видов, которые вы должны использовать в первом параметре метода. Не переживайте, если пока все это кажется странным и даже бессмысленным. Вскоре все будет понятно! Как только вы изучите несколько примеров, сразу получится стройная картина.

Вооружившись базовой информацией, не забивая голову ничем лишним, перейдем к практическим разделам. В качестве зарядки поупражняемся немного с ограничениями. Готовы? Поехали!

# 3.1. Размещение компонентов пользовательского интерфейса в центре экрана

## Постановка задачи

Требуется поместить компонент пользовательского интерфейса в центре экрана. Иными словами, мы собираемся расположить вид в центре его вышестоящего вида с помощью ограничений.

## Решение

Создайте два ограничения: одно для выравнивания позиции `center.x` целевого вида по позиции `center.x` вышестоящего вида, другое — для выравнивания позиции `center.y` целевого вида по позиции `center.y` вышестоящего вида.

## Обсуждение

Начнем с создания простой кнопки, которую выровняем по центру экрана. Как было указано в подразделе «Решение» текущего раздела, для этого всего лишь требуется гарантировать, что координаты *x* и *y* центра нашей кнопки будут соответствовать координатам *x* и *y* центра того вида, в котором находится кнопка. Для этого мы напишем два ограничения и добавим их к виду, включающему нашу кнопку (вышестоящему виду этой кнопки). Вот простой код, позволяющий добиться такого эффекта:

```
#import "ViewController.h"

@interface ViewController ()
@property (nonatomic, strong) UIButton *button;
@end

@implementation ViewController

- (void)viewDidLoad{
  [super viewDidLoad];

  /* 1) Создаем кнопку */
  self.button = [UIButton buttonWithType:UIButtonTypeSystem];
  self.button.translatesAutoresizingMaskIntoConstraints = NO;
  [self.button setTitle:@"Button" forState:UIControlStateNormal];
  [self.view addSubview:self.button];

  UIView *superview = self.button.superview;

  /* 2) Создаем ограничение для центрирования кнопки по горизонтали */
  NSLayoutConstraint *centerXConstraint =
  [NSLayoutConstraint constraintWithItem:self.button
                               attribute:NSLayoutAttributeCenterX
                               relatedBy:NSLayoutRelationEqual
                                  toItem:superview
                               attribute:NSLayoutAttributeCenterX
                              multiplier:1.0f
                                constant:0.0f];

  /* 3) Создаем ограничение для центрирования кнопки по вертикали */
  NSLayoutConstraint *centerYConstraint =
  [NSLayoutConstraint constraintWithItem:self.button
                               attribute:NSLayoutAttributeCenterY
                               relatedBy:NSLayoutRelationEqual
                                  toItem:superview
                               attribute:NSLayoutAttributeCenterY
                              multiplier:1.0f
                                constant:0.0f];

  /* Добавляем ограничения к вышестоящему виду кнопки */
```

```
    [superview addConstraints:@[centerXConstraint, centerYConstraint]];
}
@end
```

Этот контроллер вида пытается сообщить iOS, что он поддерживает все возможные ориентации интерфейса, применимые на этом устройстве. Этот факт подтверждает, что кнопка действительно будет расположена в центре экрана, независимо от типа устройства и его ориентации. Тем не менее, прежде чем этот метод начнет действовать, вы должны убедиться, что активировали все необходимые виды ориентации внутри самого проекта. Для этого перейдите в Xcode к свойствам целевого проекта, откройте вкладку General (Общие), а в ней найдите раздел Device Orientation (Ориентация устройства). Затем активизируйте все возможные виды ориентации (рис. 3.2).

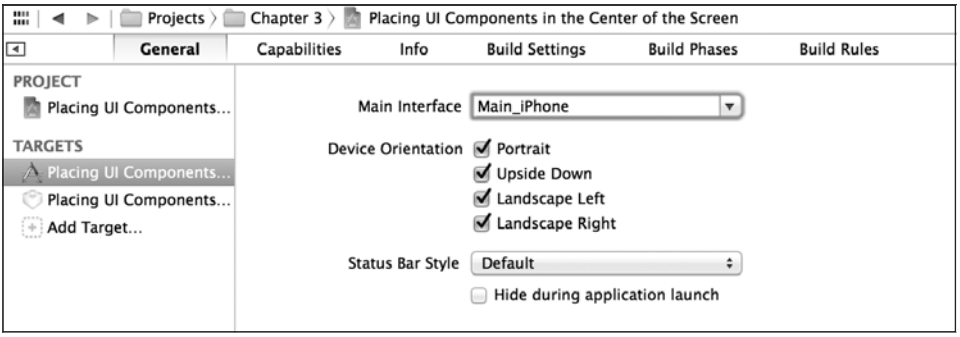


**Рис. 3.2.** Активизируем в Xcode все виды ориентации, поддерживаемые для целевого проекта

Теперь, если запустить это приложение на устройстве или эмуляторе, вы увидите на экране обычную кнопку. Сколько бы вы ни вращали устройство, кнопка никуда не сдвигается с центра экрана. Мы смогли достичь этого, не написав ни строки кода для настройки фрейма кнопки, а также без прослушивания каких-либо изменений ориентации и без корректирования положения кнопки. Фактически здесь были применены только возможности автоматической компоновки (рис. 3.3). Этот подход выигрышен по той простой причине, что наш код теперь будет работать на любом устройстве, независимо от его ориентации и разрешения экрана. Напротив, если бы мы программировали фрейм для компонентов пользовательского интерфейса, то пришлось бы создавать отдельные фреймы для каждого целевого устройства во всех интересующих нас ориентациях, поскольку на разных устройствах с iOS могут использоваться экраны с довольно несхожими разрешениями. В частности, приложение, написанное в этом разделе, будет отлично работать и на iPad, и на iPhone, причем кнопка будет находиться в центре экрана независимо от ориентации устройства и разрешения его экрана.

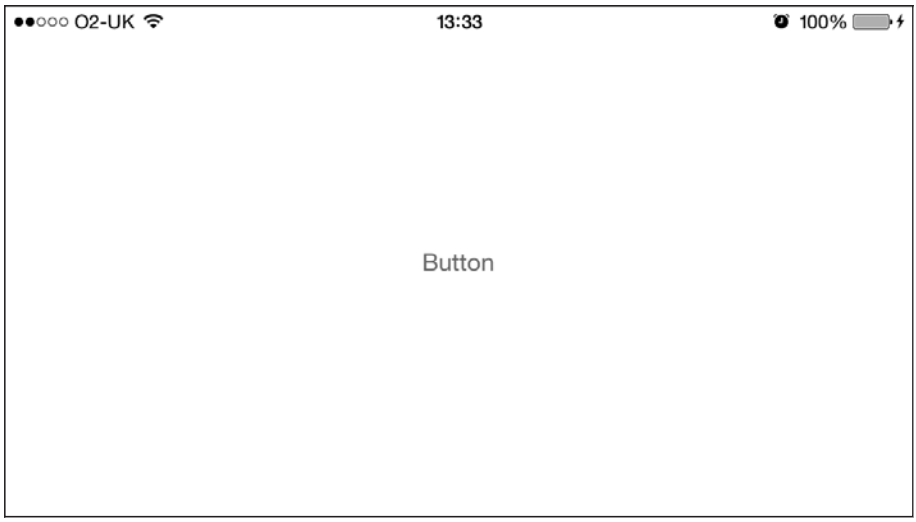


**Рис. 3.3.** Кнопка остается в центре экрана при любой ориентации

## См. также

Разделы 3.0 и 3.2.

# 3.2. Определение горизонтальных и вертикальных ограничений на языке визуального форматирования

## Постановка задачи

Требуется возможность определять ограничения, изменяющие способ выравнивания элемента пользовательского интерфейса по горизонтали или по вертикали в его вышестоящем виде.

## Решение

В строке форматирования ограничения пользуйтесь указателем ориентации `H:`, чтобы задать выравнивание по горизонтали, и указателем `V:` для выравнивания по вертикали.

## Обсуждение

Я не буду утверждать, что язык визуального форматирования прост для понимания, — напротив, он довольно запутан. Поэтому приведу несколько примеров

работы с ним, которые, надеюсь, прояснят ситуацию. Во всех этих примерах мы будем изменять горизонтальное выравнивание кнопки на экране.

1. Кнопка должна находиться на расстоянии 100 точек от каждого из краев ее вышестоящего вида:

```
H:|-100-[_button]-100-|
```

2. Кнопка должна находиться на расстоянии 100 точек или менее от левого края вышестоящего вида. Кроме того, ее ширина должна быть не меньше 50 точек, а расстояние между кнопкой и правым краем вышестоящего вида должно составлять 100 точек или менее:

```
H:|-(<=100)-[_button(>=50)]-(<=100)-|
```

3. Кнопка должна находиться на стандартном расстоянии от левого края вышестоящего вида (стандартные расстояния определяются Apple) и иметь ширину не менее 100, но не более 200 точек:

```
H:|-[_button(>=100,<=200)]
```

Как видите, может понадобиться некоторое время, чтобы привыкнуть к правилам форматирования. Но, как только вы усвоите основы этого процесса, он постепенно начнет укладываться у вас в голове. Аналогичные правила применяются и к выравниванию по вертикали, при котором используется указатель ориентации `V:`, например:

```
V:[_button]-(>=100)-|
```

При таком ограничении кнопка «прилипнет» к верхнему краю вышестоящего вида (не забывайте, что это ограничение действует по вертикали, так как начинается с `V`) и будет находиться на расстоянии не менее 100 точек от его нижнего края.

Итак, опробуем изученный материал на практике. Напишем на языке визуального форматирования ограничения, позволяющие сделать примерно такой же интерфейс, как на рис. 3.4.

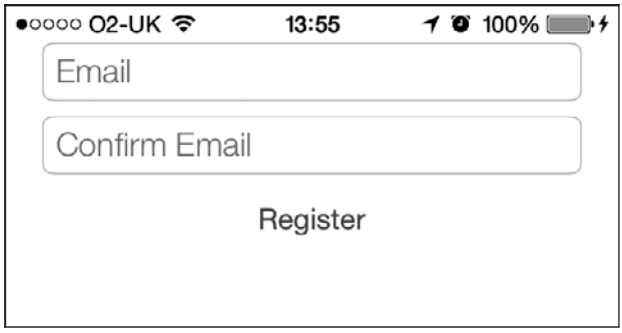


**Рис. 3.4.** Интерфейс, который мы хотим получить, опираясь на наши ограничения и пользуясь языком визуального форматирования

Чтобы дизайнерам было проще принимать решения, а сами приложения выглядели более единообразно, Apple регламентирует стандартные расстояния (пробелы), которые следует оставлять между компонентами пользовательского интерфейса. Эти стандарты описаны в документе iOS Human Interface Guidelines.

Прежде чем вплотную заняться написанием кода, опишем ограничения, которые необходимо реализовать в соответствии с предыдущим рисунком.

- Поле для адреса электронной почты имеет стандартное расстояние по вертикали до верхней границы вида.
- Поле для подтверждения адреса электронной почты имеет стандартное расстояние по вертикали до поля с адресом электронной почты.
- Кнопка Register (Зарегистрировать) имеет стандартное расстояние по вертикали до поля для подтверждения адреса электронной почты.
- Все компоненты центрированы по горизонтали относительно родительского (вышестоящего) вида.
- Поля для адреса электронной почты и подтверждения этого адреса имеют стандартное расстояние по горизонтали от левого и правого краев вышестоящего вида.
- Ширина кнопки является фиксированной и составляет 128 точек.

Рассмотрим код, необходимый для реализации всех этих требований. Для начала просто определим все ограничения на языке визуального форматирования и поместим эти определения выше вида с контроллером:

```
/* Ограничения для поля с адресом электронной почты */
NSString *const kEmailTextFieldHorizontal = @"H:|-[_textFieldEmail]-|";
NSString *const kEmailTextFieldVertical = @"V:|-[_textFieldEmail]";

/* Ограничения для поля, в котором подтверждается адрес электронной почты */
NSString *const kConfirmEmailHorizontal = @"H:|-[_textFieldConfirmEmail]-|";
NSString *const kConfirmEmailVertical =
@"V:[_textFieldEmail]-[_textFieldConfirmEmail]";

/* Ограничение для регистрационной кнопки */
NSString *const kRegisterVertical =
@"V:[_textFieldConfirmEmail]-[_registerButton]";
```

Здесь мы видим, что оба текстовых поля сопровождаются применяемыми к ним по горизонтали и вертикали ограничениями, описанными на языке визуального форматирования. Кнопка регистрации, в свою очередь, имеет только ограничение по вертикали, также описанное на языке визуального форматирования. Почему? Оказывается, что на языке визуального форматирования невозможно выразить центрирование компонента пользовательского интерфейса по горизонтали. Для решения этой задачи воспользуемся приемом, изученным в разделе 3.1. Но пусть это вас не смущает — все равно стоит пользоваться языком визуального форматирования и наслаждаться его потенциалом. Да, он несовершенен, но это не повод от него отказываться.

Теперь определим компоненты пользовательского интерфейса как закрытые (приватные) свойства в файле реализации контроллера вида:

```
@interface ViewController ()
@property (nonatomic, strong) UITextField *textFieldEmail;
@property (nonatomic, strong) UITextField *textFieldConfirmEmail;
@property (nonatomic, strong) UIButton *registerButton;
@end

@implementation ViewController

<# Оставшаяся часть вашего кода находится здесь #>
```

Что дальше? Теперь нужно сконструировать сами компоненты пользовательского интерфейса в файле реализации контроллера вида. Итак, напишем два удобных метода, которые нам в этом помогут. Опять же не забывайте: мы не собираемся здесь задавать фреймы этих компонентов. Позже нам в этом поможет автоматическая компоновка:

```
- (UITextField *) textFieldWithPlaceholder:(NSString *)paramPlaceholder{

  UITextField *result = [[UITextField alloc] init];
  result.translatesAutoresizingMaskIntoConstraints = NO;
  result.borderStyle = UITextBorderStyleRoundedRect;
  result.placeholder = paramPlaceholder;
  return result;

}

- (void) constructUIComponents{

  self.textFieldEmail =
  [self textFieldWithPlaceholder:@"Email"];

  self.textFieldConfirmEmail =
  [self textFieldWithPlaceholder:@"Confirm Email"];

  self.registerButton = [UIButton buttonWithType:UIButtonTypeSystem];
  self.registerButton.translatesAutoresizingMaskIntoConstraints = NO;
  [self.registerButton setTitle:@"Register" forState:UIControlStateNormal];

}
```

Метод `textFieldWithPlaceholder`: просто создает текстовые поля, содержащие заданный подстановочный текст, а метод `constructUIComponents`, в свою очередь, создает два текстовых поля, пользуясь вышеупомянутым методом и кнопкой. Вы, вероятно, заметили, что мы присвоили свойству `translatesAutoresizingMaskIntoConstraints` всех наших компонентов пользовательского интерфейса значение `NO`. Так мы помогаем UIKit не перепутать маски автоматической подгонки размеров с ограничениями автоматической компоновки. Как вы знаете, можно задавать маски автоматической подгонки размеров для компонентов пользовательского интерфейса и контроллеров

видов как в коде, так и в конструкторе интерфейсов. Об этом мы говорили в главе 1. Устанавливая здесь значение NO, мы гарантируем, что UIKit ничего не перепутает и не будет автоматически преобразовывать маски автоматической подгонки размера в ограничения автоматической компоновки. Эту функцию необходимо задавать, если вы смешиваете свойства автоматической компоновки компонентов с ограничениями макета. Как правило, следует устанавливать это значение у всех компонентов пользовательского интерфейса в NO всякий раз, когда вы работаете с ограничениями автоматической компоновки. Исключение составляют случаи, в которых вы специально приказываете UIKit преобразовать маски автоматической подгонки размеров в ограничения автоматической компоновки.

Мы создаем компоненты пользовательского интерфейса, но вполне очевидно, что методу viewDidLoad контроллера вида необходимо добавить к виду все три компонента пользовательского интерфейса. Почему бы не написать еще один небольшой метод, который будет заниматься именно этим?

```
- (void) addUIComponentsToView:(UIView *)paramView{

  [paramView addSubview:self.textFieldEmail];
  [paramView addSubview:self.textFieldConfirmEmail];
  [paramView addSubview:self.registerButton];

}
```

Итак, почти все готово. Следующая крупная задача — создать методы, которые позволят сконструировать и собрать все ограничения в массив. У нас также есть удобный четвертый метод, который собирает все ограничения от всех трех компонентов пользовательского интерфейса и объединяет их в общий большой массив. Вот как мы его реализуем:

```
- (NSArray *) emailTextFieldConstraints{

  NSMutableArray *result = [[NSMutableArray alloc] init];

  NSDictionary *viewsDictionary =
  NSDictionaryOfVariableBindings(_textFieldEmail);

  [result addObjectsFromArray:
   [NSLayoutConstraint constraintsWithVisualFormat:kEmailTextFieldHorizontal
                                           options:0
                                           metrics:nil
                                             views:viewsDictionary]
  ];

  [result addObjectsFromArray:
   [NSLayoutConstraint constraintsWithVisualFormat:kEmailTextFieldVertical
                                           options:0
                                           metrics:nil
                                             views:viewsDictionary]
  ];
```

```
  return [NSArray arrayWithArray:result];

}

- (NSArray *) confirmEmailTextFieldConstraints{

  NSMutableArray *result = [[NSMutableArray alloc] init];
  NSDictionary *viewsDictionary =
  NSDictionaryOfVariableBindings(_textFieldConfirmEmail, _textFieldEmail);

  [result addObjectsFromArray:
   [NSLayoutConstraint constraintsWithVisualFormat:kConfirmEmailHorizontal
                                           options:0
                                           metrics:nil
                                             views:viewsDictionary]
  ];

  [result addObjectsFromArray:
   [NSLayoutConstraint constraintsWithVisualFormat:kConfirmEmailVertical
                                           options:0
                                           metrics:nil
                                             views:viewsDictionary]
  ];

  return [NSArray arrayWithArray:result];

}

- (NSArray *) registerButtonConstraints{

  NSMutableArray *result = [[NSMutableArray alloc] init];

  NSDictionary *viewsDictionary =
  NSDictionaryOfVariableBindings(_registerButton, _textFieldConfirmEmail);

  [result addObject:

   [NSLayoutConstraint constraintWithItem:self.registerButton
                                attribute:NSLayoutAttributeCenterX
                                relatedBy:NSLayoutRelationEqual
                                   toItem:self.view
                                attribute:NSLayoutAttributeCenterX
                               multiplier:1.0f
                                 constant:0.0f]

  ];

  [result addObjectsFromArray:
   [NSLayoutConstraint constraintsWithVisualFormat:kRegisterVertical
                                           options:0
                                           metrics:nil
```

```
                                                views:viewsDictionary]

  ];

  return [NSArray arrayWithArray:result];
}

- (NSArray *)constraints{
  NSMutableArray *result = [[NSMutableArray alloc] init];
  [result addObjectsFromArray:[self emailTextFieldConstraints]];
  [result addObjectsFromArray:[self confirmEmailTextFieldConstraints]];
  [result addObjectsFromArray:[self registerButtonConstraints]];
  return [NSArray arrayWithArray:result];
}
```

Фактически здесь мы имеем метод экземпляра `constraints`, относящийся к контроллеру вида; этот метод собирает ограничения от всех трех компонентов пользовательского интерфейса, а потом возвращает их все как один большой массив. Теперь переходим к основной части контроллера — методу `viewDidLoad`:

```
- (void)viewDidLoad{

  [super viewDidLoad];

  [self constructUIComponents];
  [self addUIComponentsToView:self.view];
  [self.view addConstraints:[self constraints]];
}
```

Этот метод просто собирает пользовательский интерфейс, добавляя сам к себе все компоненты пользовательского интерфейса и связанные с ними ограничения. При этом он использует методы, написанные нами ранее. Отлично, но что мы увидим на экране, когда запустим эту программу? Мы уже видели, как этот интерфейс выглядит на устройстве, работающем в книжной ориентации (см. рис. 3.4). А теперь повернем устройство и посмотрим, что получится при альбомной ориентации (рис. 3.5).


**Рис. 3.5.** Ограничения функционируют в альбомном режиме не хуже, чем в книжном

## См. также

Разделы 3.0 и 3.1.

# 3.3. Применение ограничений при работе с перекрестными видами

## Постановка задачи

Требуется выровнять компонент пользовательского интерфейса относительно другого компонента пользовательского интерфейса, притом что родительские элементы у этих компонентов разные.

## Решение

Ориентируясь на рис. 3.1, убедитесь, что вам удалось найти ближайший общий вышестоящий вид, являющийся родителем для интересующих вас компонентов пользовательского интерфейса. Затем добавьте ограничения к этому вышестоящему виду.

## Обсуждение

Прежде чем углубляться в детали, разберемся, в чем же заключаются ограничения перекрестных видов. Мне кажется, что суть проблемы удобнее изобразить на картинке, а не описывать словами, — предлагаю вашему вниманию рис. 3.6.

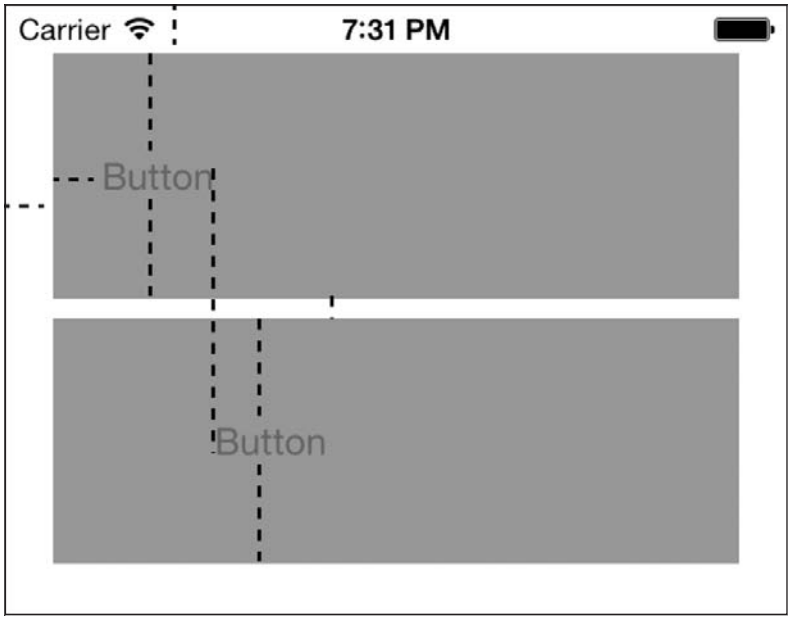


**Рис. 3.6.** Важные ограничения, налагаемые перекрестными видами на две кнопки

На этом рисунке к видам применяется немало ограничений. Разберем их по порядку, разложив все по полочкам.

- Есть основной вид с контроллером, в этом виде расположены еще два серых вида. Оба они должны отстоять от левой и правой границ вида с контроллером на стандартные расстояния. В частности, должно сохраняться стандартное расстояние между верхним серым видом и верхней границей вышестоящего вида. Между двумя серыми видами по вертикали также должно сохраняться стандартное пространство.
- Нужна кнопка, которая будет вертикально центрирована относительно обоих серых видов.
- Кнопка, расположенная в верхнем сером виде, слева должна быть удалена от края своего вышестоящего вида на стандартное расстояние.
- Левая сторона кнопки, находящейся в нижнем сером виде, должна быть выровнена по правой стороне той кнопки, что находится в верхнем сером виде. Это и есть ограничение для перекрестных видов, которое интересует нас в данном разделе.
- Серые виды должны автоматически изменять размер по мере того, как меняется ориентация вида с контроллером.
- Высота обоих серых видов должна составлять по 100 точек.

Итак, начнем. Чтобы выполнить все перечисленные задачи, вначале обратимся к методу `viewDidLoad` контроллера вида. Всегда стоит продумывать максимально чистый способ объединения методов. Конечно, в данном примере мы оперируем довольно большим количеством ограничений и видов. Как же нам не захламлять метод `viewDidLoad` контроллера вида? Вот так:

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

  [self createButtons];

  [self applyConstraintsToTopGrayView];
  [self applyConstraintsToButtonOnTopGrayView];

  [self applyConstraintsToBottomGrayView];
  [self applyConstraintsToButtonOnBottomGrayView];

}
```

Мы просто распределили стоящие перед нами задачи по разным методам, которые вскоре реализуем. Продолжим — определим виды в файле реализации контроллера вида как расширение интерфейса:

```
#import "ViewController.h"

@interface ViewController ()
@property (nonatomic, strong) UIView *topGrayView;
```

```
@property (nonatomic, strong) UIButton *topButton;
@property (nonatomic, strong) UIView *bottomGrayView;
@property (nonatomic, strong) UIButton *bottomButton;
@end
@implementation ViewController

<# Оставшаяся часть вашего кода находится здесь #>
```

Далее следует реализовать метод `createGrayViews`. Как понятно из названия, этот метод отвечает за создание серых видов:

```
- (UIView *) newGrayView{

  UIView *result = [[UIView alloc] init];
  result.backgroundColor = [UIColor lightGrayColor];
  result.translatesAutoresizingMaskIntoConstraints = NO;
  [self.view addSubview:result];
  return result;

}

- (void) createGrayViews{

  self.topGrayView = [self newGrayView];
  self.bottomGrayView = [self newGrayView];
}
```

Пока несложно? Оба серых вида добавляются к контроллеру нашего вида. Отлично. Что дальше? Теперь нужно реализовать метод `createButtons`, поскольку он вызывается в методе `viewDidLoad` контроллера вида. Этот метод должен просто создать кнопки и поместить каждую в ассоциированном с ней сером виде:

```
- (UIButton *) newButtonPlacedOnView:(UIView *)paramView{

  UIButton *result = [UIButton buttonWithType:UIButtonTypeSystem];
  result.translatesAutoresizingMaskIntoConstraints = NO;
  [result setTitle:@"Button" forState:UIControlStateNormal];
  [paramView addSubview:result];
  return result;

}

- (void) createButtons{
  self.topButton = [self newButtonPlacedOnView:self.topGrayView];
  self.bottomButton = [self newButtonPlacedOnView:self.bottomGrayView];
}
```

Опять же в методе `createButtons` мы видим, что после создания серых видов и кнопок нужно применить ограничения к этим видам и кнопкам. Начнем с применения ограничений к верхнему серому виду. Эти ограничения должны обеспечивать соблюдение следующих условий:

- верхний вид должен находиться на стандартном расстоянии от вида с контроллером по левому и верхнему краю;
- высота этого серого вида должна составлять 100 точек.

```
- (void) applyConstraintsToTopGrayView{

  NSDictionary *views =

  NSDictionaryOfVariableBindings(_topGrayView);

  NSMutableArray *constraints = [[NSMutableArray alloc] init];

  NSString *const kHConstraint = @"H:|-[_topGrayView]-|";
  NSString *const kVConstraint = @"V:|-[_topGrayView(==100)]";

  /* Горизонтальные ограничения */
  [constraints addObjectsFromArray:
   [NSLayoutConstraint constraintsWithVisualFormat:kHConstraint
                                           options:0
                                           metrics:nil
                                             views:views]
  ];

  /* Вертикальные ограничения */
  [constraints addObjectsFromArray:
   [NSLayoutConstraint constraintsWithVisualFormat:kVConstraint
                                           options:0
                                           metrics:nil
                                             views:views]
  ];

  [self.topGrayView.superview addConstraints:constraints];

}
```

Здесь следует остановиться на том, как создается вертикальное ограничение верхнего серого вида. Как видите, мы задаем высоту верхнего вида равной 100 точкам и записываем эту информацию в формате (`==100`). Среда времени исполнения интерпретирует это значение именно как высоту, поскольку здесь есть указатель `V:`. Он сообщает среде времени исполнения о следующем: те числа, которые мы сообщаем системе, как-то связаны с высотой и вертикальным выравниванием целевого вида, а не с его шириной и горизонтальным выравниванием.

Далее займемся установкой ограничений для кнопки, находящейся в верхнем сером виде. Это делается с помощью метода `applyConstraintsToButtonOnTopGrayView`. Кнопка должна будет соответствовать перечисленным далее ограничениям:

- она должна быть вертикально центрирована в верхнем сером виде;
- она должна быть удалена на стандартное расстояние от левого и верхнего края этого серого вида.

У нее не должно быть жестко заданных высоты и ширины; эти значения будут зависеть от содержимого кнопки, в данном случае — от текста Button, который мы решили на ней написать:

```
- (void) applyConstraintsToButtonOnTopGrayView{
  NSDictionary *views = NSDictionaryOfVariableBindings(_topButton);

  NSMutableArray *constraints = [[NSMutableArray alloc] init];

  NSString *const kHConstraint = @"H:|-[_topButton]";

  /* Горизонтальные ограничения */
  [constraints addObjectsFromArray:
   [NSLayoutConstraint constraintsWithVisualFormat:kHConstraint
                                           options:0
                                           metrics:nil
                                             views:views]
  ];

  /* Вертикальные ограничения */
  [constraints addObject:
   [NSLayoutConstraint constraintWithItem:self.topButton
                                attribute:NSLayoutAttributeCenterY
                                relatedBy:NSLayoutRelationEqual
                                   toItem:self.topGrayView
                                attribute:NSLayoutAttributeCenterY
                               multiplier:1.0f
                                 constant:0.0f]
  ];

  [self.topButton.superview addConstraints:constraints];

}
```

Итак, работа с верхним серым видом и находящейся в нем кнопкой завершена. Переходим к нижнему серому виду и его кнопке. Сейчас начнем работать с методом ConstraintsToBottomGrayView. Он будет задавать ограничения для нижнего серого вида. Просто напомню, что для этого вида нам требуется создать следующие ограничения:

- вид удален на стандартное расстояние от верхнего и левого края вышестоящего вида с контроллером;
- вид удален на стандартное расстояние от нижней границы верхнего серого вида;
- высота нижнего серого вида составляет 100 точек.

```
- (void) applyConstraintsToBottomGrayView{

  NSDictionary *views =
  NSDictionaryOfVariableBindings(_topGrayView,
                                 _bottomGrayView);

  NSMutableArray *constraints = [[NSMutableArray alloc] init];
```

```
  NSString *const kHConstraint = @"H:|-[_bottomGrayView]-|";
  NSString *const kVConstraint =
  @"V:|-[_topGrayView]-[_bottomGrayView(==100)]";

  /* Горизонтальные ограничения */
  [constraints addObjectsFromArray:
   [NSLayoutConstraint constraintsWithVisualFormat:kHConstraint
                                           options:0
                                           metrics:nil
                                             views:views]
  ];

  /* Вертикальные ограничения */
  [constraints addObjectsFromArray:
   [NSLayoutConstraint constraintsWithVisualFormat:kVConstraint
                                           options:0
                                           metrics:nil
                                             views:views]
  ];

  [self.bottomGrayView.superview addConstraints:constraints];

}
```

Вертикальные ограничения для нижнего серого вида, выраженные на языке визуального форматирования, выглядят длинновато, но, в сущности, они тривиальны. Приглядевшись к ним повнимательнее, вы заметите, что эти ограничения просто выравнивают верхний и нижний серые виды по сторонам их общего вышестоящего вида с контроллером. При этом используются указатели стандартного расстояния и постоянная высота, равная 100 точкам.

Следующий и, пожалуй, последний компонент пользовательского интерфейса, для которого мы собираемся написать ограничения, — это кнопка, расположенная в нижнем сером виде. Метод, который будет заниматься ее ограничениями, называется `applyConstraintsToButtonOnBottomGrayView`. Перед тем как его реализовать, обсудим требования, которым должны соответствовать ограничения для нижней кнопки:

- кнопка должна быть вертикально центрирована в нижнем сером виде;
- ее левый край должен быть выровнен по правому краю кнопки, находящейся в верхнем сером виде;
- с ней не должны применяться строго определенные значения высоты и ширины; ее высота и ширина должны зависеть от содержимого — в данном случае от текста `Button`, который мы на ней записываем.

```
- (void) applyConstraintsToButtonOnBottomGrayView{

  NSDictionary *views = NSDictionaryOfVariableBindings(_topButton,
                                                       _bottomButton);

  NSString *const kHConstraint = @"H:[_topButton][_bottomButton]";
```

```
  /* Горизонтальные ограничения */
  [self.bottomGrayView.superview addConstraints:
   [NSLayoutConstraint constraintsWithVisualFormat:kHConstraint
                                           options:0
                                           metrics:nil
                                             views:views]
  ];
  /* Вертикальные ограничения */
  [self.bottomButton.superview addConstraint:
   [NSLayoutConstraint constraintWithItem:self.bottomButton
                                attribute:NSLayoutAttributeCenterY
                                relatedBy:NSLayoutRelationEqual
                                   toItem:self.bottomGrayView
                                attribute:NSLayoutAttributeCenterY
                               multiplier:1.0f
                                 constant:0.0f]
  ];

}
```

Наконец, мы должны удостовериться в том, что контроллер вида сообщает среде времени исполнения, что он может обрабатывать любые варианты ориентации. Ведь именно этот аспект наиболее сильно интересовал нас в данном разделе. Поэтому мы переопределим метод `supportedInterfaceOrientations` в виде `UIViewController`:

```
- (NSUInteger) supportedInterfaceOrientations{
  return UIInterfaceOrientationMaskAll;
}
```

Итак, работа с этим контроллером вида завершена. Запустим приложение и посмотрим, как оно работает при книжной ориентации (рис. 3.7).

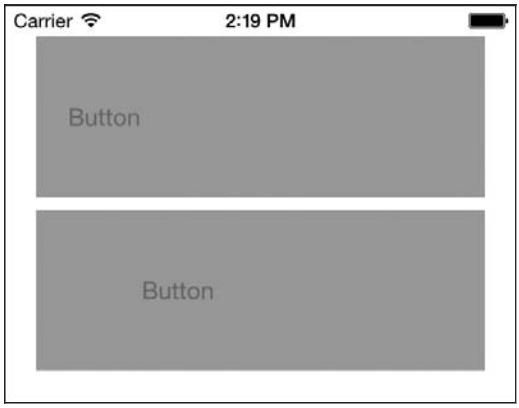


**Рис. 3.7.** Приложение отображает компоненты пользовательского интерфейса в книжной ориентации согласно требованиям, которые мы предъявили

А теперь момент истины! Будет ли оно работать в альбомном режиме? Попробуем (рис. 3.8).

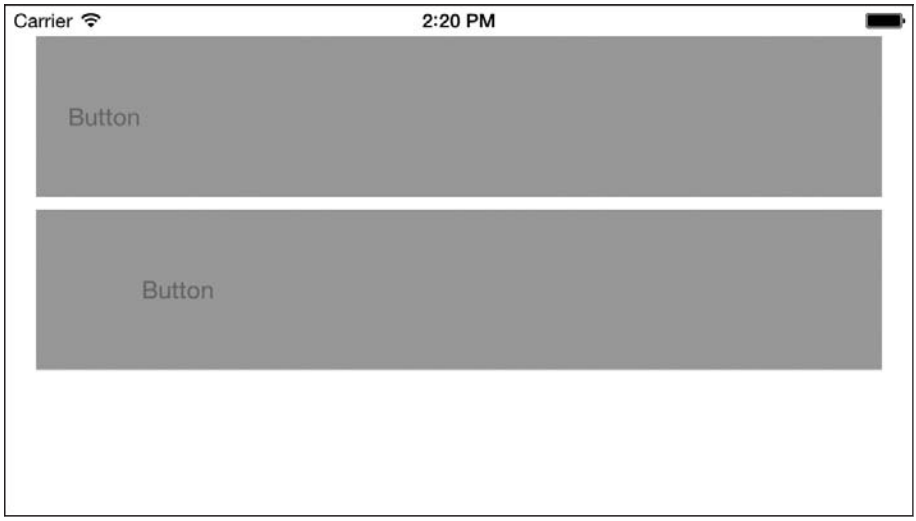


**Рис. 3.8.** Как и ожидалось, тот же самый код отлично работает и при альбомной ориентации экрана

Отлично! Все получилось.

## См. также

Раздел 3.0.

# 3.4. Конфигурирование ограничений автоматической компоновки в конструкторе интерфейсов

## Постановка задачи

Требуется задействовать весь потенциал конструктора интерфейсов для создания ограничений при работе с пользовательским интерфейсом.

## Решение

Выполните следующие шаги.

1. Откройте в конструкторе интерфейсов файл XIB или файл раскадровки, который вы собираетесь редактировать.

2. Убедитесь, что в конструкторе интерфейсов вы выбрали объект вида, в котором собираетесь активизировать автоматическую компоновку. Просто щелкните на этом объекте.
3. Щелкните на элементе меню View ▸ Utilities ▸ Show File Inspector (Вид ▸ Утилиты ▸ Показать инспектор файлов).
4. Убедитесь, что в элементе File Inspector (Инспектор файлов) в разделе Interface Builder Document (Документ конструктора интерфейсов) установлен флажок Use Autolayout (Использовать автоматическую компоновку) (рис. 3.9).

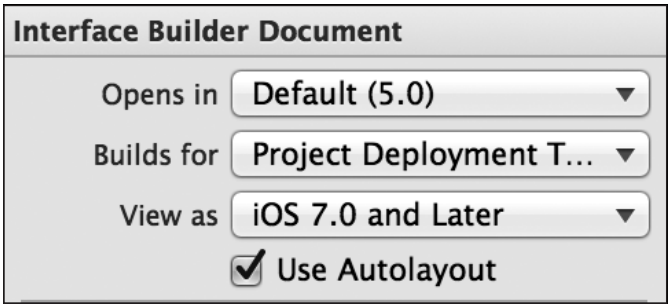


**Рис. 3.9.** Активизируем автоматическую компоновку в конструкторе интерфейсов

## Обсуждение

Конструктор интерфейсов значительно упрощает для программиста создание ограничений, причем наше участие в этом сводится к минимуму. До того как в iOS появилась возможность автоматической компоновки, приходилось, как правило, пользоваться специальными ориентировочными панелями (guideline bars). Эти панели появлялись на экране, пока вы перемещали компоненты пользовательского интерфейса. Ориентировочные панели были связаны с масками для автоматической подгонки размеров, которые вы могли создавать и в коде, точно так же, как ограничения. Но после того, как в конструкторе интерфейсов будет установлен флажок Use Autolayout (Использовать автоматическую компоновку), ориентировочные панели приобретут несколько иное значение. Теперь они сообщают о тех ограничениях, которые создает для вас в фоновом режиме сам конструктор интерфейсов.

Немного поэкспериментируем. Создадим в Xcode приложение с одним видом (Single View Application). Таким образом, будет создано приложение, содержащее всего один контроллер вида. Этот контроллер вида будет относиться к классу `ViewController`, а `.xib`-файл для него будет называться `ViewController.xib`. Просто щелкните на этом файле, чтобы конструктор интерфейсов открыл его. Убедитесь, что в инспекторе файлов установлен флажок Use Autolayout (Использовать автоматическую компоновку) так, как описано в подразделе «Решение» этого раздела.

Теперь просто найдите в библиотеке объектов *кнопку* (Button) и перетащите ее в центр экрана. Дождитесь, пока в конструкторе интерфейсов появятся ориентировочные панели, по которым будет понятно, что центр кнопки соответствует центру экрана. В меню Edit (Правка) установите флажок Show Document Outline (По-

казать структуру документа). Если у вас в конструкторе интерфейсов уже открыт раздел Document Outline (Структура документа), то вместо Show Document Outline (Показать структуру документа) на этом месте будет отображаться надпись Hide Document Outline (Скрыть структуру документа) — в таком случае ничего делать не надо. Теперь найдите в разделе Document Outline (Структура документа) новый подраздел, отмеченный голубым цветом. Он был создан специально для вас и называется Constraints (Ограничения). Раскройте ограничения, созданные конструктором интерфейсов для этой кнопки. То, что вы теперь увидите, должно напоминать рис. 3.10.

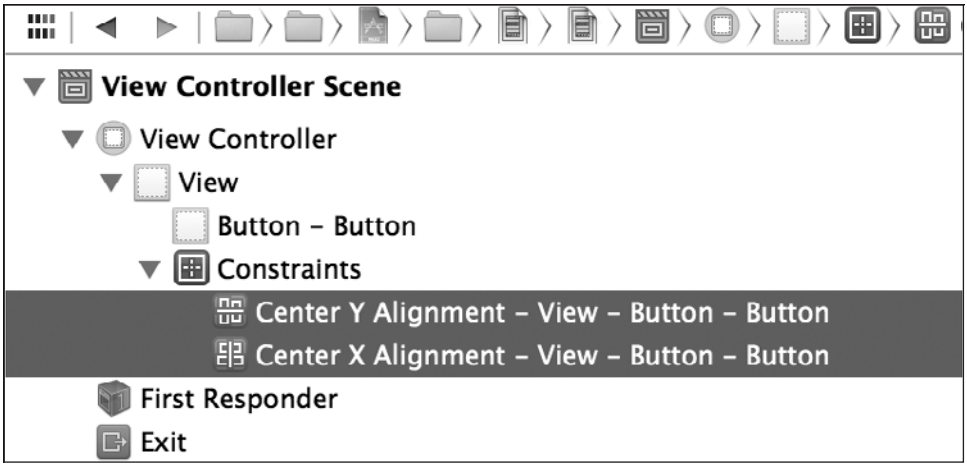


**Рис. 3.10.** Конструктор интерфейсов создал ограничения компоновки

С помощью конструктора интерфейсов вы можете создать довольно много ограничений, и для этого не потребуется писать ни единой строки кода. Но случается, что нужные вам ограничения настолько сложны, что их лучше запрограммировать в коде. Определившись с тем, как компоненты пользовательского интерфейса должны быть расположены на экране, вы поймете, как лучше поступить — сформулировать их в конструкторе интерфейса, выразить в коде или сделать и то и другое.

## См. также

Раздел 3.0.

# 4 Создание и использование табличных видов

## 4.0. Введение

Табличный вид — это обычный вид с прокручиваемым контентом, который разделен на секции. Каждая такая секция, в свою очередь, подразделяется на строки. Каждая строка (Row) является экземпляром класса `UITableViewCell`. Вы можете создавать собственные варианты строк в табличном виде, *делая подклассы* этого класса.

Табличный вид — это сущность, которая идеально подходит для представления пользователю списка элементов. В ячейки табличных видов можно встраивать изображения, текст и другие объекты. Можно самостоятельно настраивать высоту, контуры, группирование ячеек и многие другие параметры. Благодаря структурной простоте табличные виды отлично подходят для адаптации под конкретные задачи.

Табличный вид можно наполнить данными, используя источник данных табличного вида. Вы можете получать различные события и управлять оформлением табличных видов с помощью объекта-делегата табличного вида. Источник данных для табличного вида определяется в протоколе `UITableViewDataSource`, а делегат табличного вида — в протоколе `UITableViewDelegate`.

Хотя экземпляр `UITableView` является подклассом от `UIScrollView`, табличные виды можно прокручивать только по вертикали. Это скорее благо, чем ограничение. В данной главе мы обсудим различные способы создания табличных видов, их настройки и управления ими.

Табличные виды можно использовать двумя способами:

- с помощью класса `UITableViewController`. Этот класс напоминает `UIViewController` (см. раздел 1.9) в том, что фактически это контроллер вида, но в нем отображается не обычный вид, а таблица. Красота этого класса заключается в том, что каждый его экземпляр уже соответствует протоколам `UITableViewDelegate` и `UITableViewDataSource`. Итак, по умолчанию контроллер табличного вида становится источником данных и одновременно делегатом того табличного вида, которым он управляет. Таким образом, чтобы реализовать, например, источник

данных для табличного вида, вам всего лишь потребуется реализовать контроллер для табличного вида, а не устанавливать вручную контроллер вида в качестве источника данных для табличного вида;

- вручную инстанцировав класс `UITableView`.

Оба этих метода вполне допустимы для создания табличных видов. Первый метод обычно используется для создания табличного вида, который целиком заполняет свой контейнер (либо окно/экран, если данный контроллер вида является корневым контроллером вида основного окна приложения). Второй метод более удобен в ситуациях, когда вы собираетесь отобразить табличный вид в качестве небольшого компонента пользовательского интерфейса, чтобы таблица, скажем, наполовину занимала экран по ширине и/или высоте. Но вы с тем же успехом можете использовать второй метод для установки высоты и ширины табличного вида в значения высоты и ширины его объемлющего окна так, чтобы табличный вид занимал целый экран. В этой главе мы исследуем оба описанных метода.

Рассмотрим пример создания табличного вида в приложении. Примеры контроллеров табличных видов будут подробно изучены в разделе 4.9, а пока мы просто займемся созданием таких видов в коде и будем добавлять их к имеющемуся контроллеру вида.

Класс `UITableView` инстанцируется с помощью метода `initWithFrame:style:`. Далее перечислены параметры, которые мы должны передать этому методу, а также значения этих параметров.

- `initWithFrame` — это параметр типа `CGRect`. Он указывает, как именно должен быть расположен табличный вид в вышестоящем виде. Если вы хотите, чтобы таблица просто полностью накрывала вышестоящий вид, передайте этому параметру значение свойства `bounds` вида с контроллером.
- `style` — это параметр типа `UITableViewStyle`, определяемый следующим образом:

```
typedef NS_ENUM(NSInteger, UITableViewStyle) {
  UITableViewStylePlain,
  UITableViewStyleGrouped
};
```

На рис. 4.1 показана разница между обычным и сгруппированным табличными видами.

Мы заполняем табличный вид информацией, используя его источник данных, как будет показано в разделе 4.1. Табличные виды также обладают делегатами. Делегаты получают различные события от табличного вида. Объекты делегатов должны соответствовать протоколу `UITableViewDelegate`. Далее перечислены отдельные методы этого протокола, которые необходимо знать.

- `tableView:viewForHeaderInSection:` — вызывается в делегате, когда табличному виду требуется отобразить заголовочный вид раздела. Каждый раздел табличного вида может содержать верхний колонтитул, некоторое количество ячеек и нижний колонтитул. В этой главе мы подробно обсудим все эти участки таблицы. Верхний и нижний колонтитул — это обычные экземпляры `UIView`.

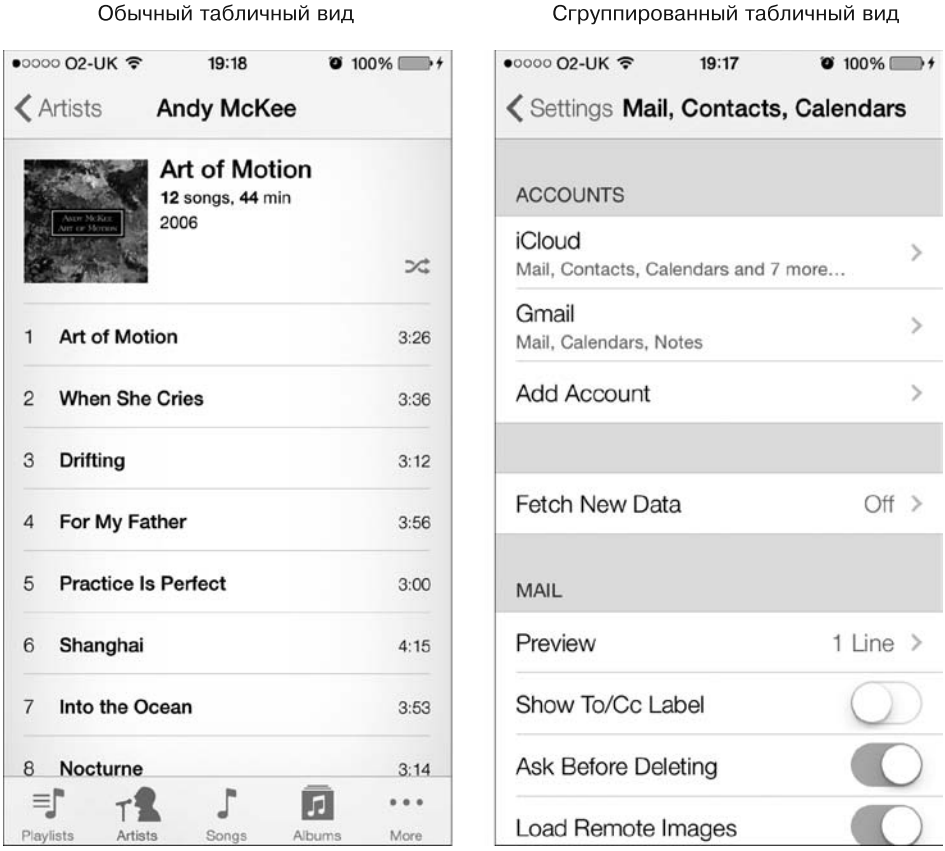


**Рис. 4.1.** Табличные виды различных типов

Данный метод является необязательным, но если вы хотите сконфигурировать заголовок для разделов вашего табличного вида, то пользуйтесь этим методом, чтобы создать экземпляр вида и передать его обратно в качестве возвращаемого значения. О верхних и нижних колонтитулах табличных видов подробнее рассказано в разделе 4.5.

- `tableView:viewForFooterInSection:` — делегатный метод, аналогичный `tableView:viewForHeaderInSection:`, но он возвращает вид с нижним колонтитулом таблицы. Как и заголовок, нижний колонтитул таблицы не является обязательным, но если он вам нужен, то его следует создавать здесь. Подробнее о верхних и нижних колонтитулах табличных видов рассказано в разделе 4.5.
- `tableView:didEndDisplayingCell:forRowAtIndexPath:` — вызывается в объекте-делегате, когда в ходе прокрутки таблицы ячейка уходит с экрана. Этот метод действительно очень удобен для вызова в делегате, так как вы можете удалять объекты и выбрасывать их из памяти, если эти объекты ассоциированы с ячейкой, которая ушла с экрана, а вы полагаете, что связанные с ней объекты вам больше не понадобятся.

- `tableView:willDisplayCell:forRowAtIndexPath:` — этот метод вызывается в делегате табличного вида всякий раз, когда ячейка вот-вот отобразится на экране.

Чтобы задать делегат для табличного вида, просто укажите в качестве значения свойства `delegate` экземпляра `UITableView` такой объект, который соответствует протоколу `UITableViewDelegate`. Если табличный вид является частью контроллера вида, то можно просто сделать этот контроллер делегатом вашего табличного вида, вот так:

```
#import "ViewController.h"

@interface ViewController () <UITableViewDelegate>
@property (nonatomic, strong) UITableView *myTableView;
@end

@implementation ViewController

- (void)viewDidLoad{
  [super viewDidLoad];

  self.myTableView = [[UITableView alloc]
                      initWithFrame:self.view.bounds
                      style:UITableViewStylePlain];

  self.myTableView.delegate = self;

  [self.view addSubview:self.myTableView];

}

@end
```

Можно считать делегат табличного вида объектом, который слушает различные события, отправляемые табличным видом. Например, такие события происходят, когда пользователь выделяет одну из ячеек в таблице либо табличному виду требуется узнать высоту всех входящих в него ячеек.

Объект-делегат обязан отвечать на сообщения, помеченные протоколом UITableViewDelegate как @required. Отвечать на другие сообщения не обязательно, но делегат должен отвечать на все сообщения, которые, по вашему замыслу, будут изменять табличный вид.

Сообщения, отправляемые объекту-делегату табличного вида, несут с собой параметр, который сообщает делегату, какой именно табличный вид инициировал данное событие в своем делегате. Это очень важно отметить, так как в определенных обстоятельствах вы можете разместить в одном объекте (как правило, в виде) более одной таблицы. Поэтому настоятельно рекомендую принимать соответствующие решения с учетом того, какой именно табличный вид послал конкретное сообщение объекту-делегату:

```
- (CGFloat) tableView:(UITableView *)tableView
  heightForRowAtIndexPath:(NSIndexPath *)indexPath{
    if ([tableView isEqual:self.myTableView]){
      return 100.0f;
    }
    return 40.0f;
  }
}
```

Расположение ячейки в табличном виде представляется индексным путем этой ячейки. *Индексный путь* — это комбинация данных о разделе и индекса строки. В данном случае индекс раздела имеет нулевую базу и указывает, к каким группе или разделу относится каждая ячейка. Индекс ячейки также имеет нулевую базу и означает положение данной ячейки в ее разделе.

# 4.1. Наполнение табличного вида данными

## Постановка задачи

Требуется наполнить табличный вид данными.

## Решение

Необходимо создать объект, соответствующий протоколу `UITableViewDataSource`, и присвоить этот объект экземпляру табличного вида. Затем, отвечая на сообщения источника данных, предоставьте информацию для вашего вида. Продолжим данный пример и объявим `.h`-файл контроллера нашего вида. Позже в коде для этого вида будет создана таблица:

```
#import "ViewController.h"

static NSString *TableViewCellIdentifier = @"MyCells";

@interface ViewController () <UITableViewDataSource>
@property (nonatomic, strong) UITableView *myTableView;
@end
```

Экземпляр `TableViewCellIdentifier` содержит идентификаторы ячеек в виде статической строковой переменной. Как вы вскоре узнаете, каждая ячейка может иметь идентификатор, и это очень помогает при повторном использовании ячеек. Пока считайте эту переменную просто уникальным идентификатором всех ячеек табличного вида — на данном этапе этого достаточно.

В методе `viewDidLoad` контроллера вида создадим табличный вид и присвоим ему контроллер вида в качестве источника данных:

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

```
  self.myTableView =
    [[UITableView alloc] initWithFrame:self.view.bounds
                                 style:UITableViewStylePlain];

  [self.myTableView registerClass:[UITableViewCell class]
           forCellReuseIdentifier:TableViewCellIdentifier];

  self.myTableView.dataSource = self;

  /* Убеждаемся, что табличный вид правильно масштабируется. */
  self.myTableView.autoresizingMask =
    UIViewAutoresizingFlexibleWidth |
    UIViewAutoresizingFlexibleHeight;

  [self.view addSubview:self.myTableView];

}
```

В этом фрагменте кода все элементарно, кроме метода `registerClass:forCellReuseIdentifier:`, который мы вызываем в экземпляре табличного вида. Что же делает этот метод? Параметр `registerClass` этого метода просто принимает имя класса, соответствующее типу объекта, который вы хотите загружать в табличном виде при отображении каждой ячейки. Все ячейки внутри табличного вида должны быть прямыми или непрямыми потомками класса `UITableViewCell`. Сам этот класс предоставляет программистам довольно широкий функционал. Но при желании этот класс можно и расширить — достаточно произвести от него подкласс, добавив к новому классу требуемый функционал. Итак, возвращаемся к параметру `registerClass` вышеупомянутого метода. Вам потребуется сообщить имя класса ячеек этому параметру, а потом передать идентификатор параметру `forCellReuseIdentifier`. Вот по какой причине мы ассоциируем классы табличного вида с идентификаторами: когда позже вы заполняете табличный вид данными, можете просто передать тот же самый идентификатор методу `dequeueReusableCellWithIdentifier:forIndexPath:` табличного вида, после чего приказать табличному виду инстанцировать ячейку таблицы, если в наличии нет ячеек, доступных для повторного использования. Все это просто отлично, так как в предыдущих версиях iOS SDK программистам приходилось инстанцировать эти ячейки самостоятельно, если из табличного вида не удавалось добыть уже готовый код, пригодный для повторного использования.

Теперь необходимо убедиться в том, что наш табличный вид реагирует на методы протокола `UITableViewDataSource`, помеченные как `@required` (обязательные). Нажмите на клавиатуре комбинацию клавиш Command+Shift+O, введите в диалоговое окно имя этого протокола, затем нажмите клавишу Enter. В результате вы увидите обязательные методы данного протокола.

Класс `UITableView` определяет свойство под названием `dataSource`. Это нетипизированный объект, который должен подчиняться протоколу `UITableViewDataSource`. Всякий раз, когда табличный вид обновляется и перезагружается с помощью метода `reloadData`, табличный вид будет вызывать в своем источнике данных различные

методы, чтобы получить информацию о тех данных, которыми вы хотите заполнить таблицу. Источник данных табличного вида может реализовывать три важных метода, два из которых являются обязательными для любого источника данных:

- `numberOfSectionsInTableView:` — позволяет источнику данных информировать табличный вид о количестве разделов, которые должны быть загружены в таблицу;
- `tableView:numberOfRowsInSection:` — сообщает контроллеру вида, сколько ячеек или строк следует загрузить в каждый раздел. Номер раздела передается источнику данных в параметре `numberOfRowsInSection`. Реализация этого метода является обязательной для объекта источника данных;
- `tableView:cellForRowAtIndexPath:` — отвечает за возвращение экземпляров класса `UITableViewCell` как строк таблицы, которыми должен заполняться табличный вид. Реализация этого метода обязательна для объекта источника данных.

Итак, продолжим и реализуем эти методы в контроллере вида один за другим. Сначала сообщим табличному виду, что мы хотим отобразить три раздела:

```
- (NSInteger)numberOfSectionsInTableView:(UITableView *)tableView{

  if ([tableView isEqual:self.myTableView]){
    return 3;
  }

  return 0;

}
```

Далее сообщим табличному виду, сколько строк хотим в нем отобразить для каждого раздела:

```
- (NSInteger)tableView:(UITableView *)tableView
  numberOfRowsInSection:(NSInteger)section{

  if ([tableView isEqual:self.myTableView]){
    switch (section){
      case 0:{
        return 3;
        break;
      }
      case 1:{
        return 5;
        break;
      }
      case 2:{
        return 8;
        break;
      }
    }
```

```
  }
  return 0;

}
```

Итак, на данный момент мы приказали табличному виду отобразить три раздела. В первом разделе три строки, во втором — пять, в третьем — восемь. Что дальше? Нужно вернуть табличному виду экземпляры `UITableViewCell` — тех ячеек, которые мы хотим отобразить в таблице:

```
- (UITableViewCell *) tableView:(UITableView *)tableView
         cellForRowAtIndexPath:(NSIndexPath *)indexPath{

  UITableViewCell *result = nil;

  if ([tableView isEqual:self.myTableView]){

    cell = [tableView
            dequeueReusableCellWithIdentifier:TableViewCellIdentifier
            forIndexPath:indexPath];

    cell.textLabel.text = [NSString stringWithFormat:
                           @"Section %ld, Cell %ld",
                           (long)indexPath.section,
                           (long)indexPath.row];

  }

  return cell;

}
```

Теперь, если запустить приложение в эмуляторе iPhone, мы увидим результат работы (рис. 4.2).

Когда табличный вид перезагружается или обновляется, он запрашивает источник данных через протокол `UITableViewDataSource`, требуя у источника данных различную информацию. В первую очередь он запросит количество разделов. Каждый раздел должен содержать строки или ячейки.

После того как источник данных укажет количество разделов, табличный вид запросит количество строк, которые должны быть загружены в каждый из разделов. Источник данных получает индекс с нулевой базой для каждого раздела и на базе этого индекса решает, сколько ячеек загрузить в каждый раздел.

Табличный вид, определив количество ячеек в разделах, продолжит запрашивать источник данных о видах — один такой вид соответствует каждой ячейке того или иного раздела. Вы можете выделять экземпляры класса `UITableViewCell` и возвращать их табличному виду. Разумеется, есть свойства, которые можно задать для каждой ячейки. Это, в частности, заголовок, подзаголовок и цвет ячейки.

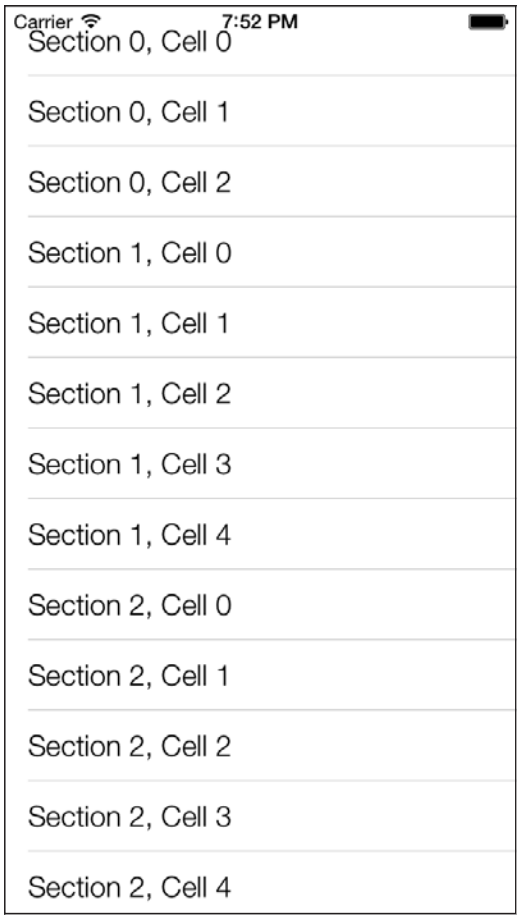


**Рис. 4.2.** Обычный табличный вид с тремя разделами

# 4.2. Использование дополнительных элементов в ячейке табличного вида

## Постановка задачи

Требуется привлечь внимание пользователя, отображая в таблице дополнительные элементы, и предложить альтернативные способы взаимодействия с каждой ячейкой в табличном виде.

## Решение

Используйте свойство `accessoryType` класса `UITableViewCell`. Экземпляры этого класса вы предоставляете табличному виду в объекте его источника данных:

```
- (UITableViewCell *) tableView:(UITableView *)tableView
         cellForRowAtIndexPath:(NSIndexPath *)indexPath{

  UITableViewCell* result = nil;

  if ([tableView isEqual:self.myTableView]){

    result = [tableView
              dequeueReusableCellWithIdentifier:MyCellIdentifier
              forIndexPath:indexPath];

    result.textLabel.text =
    [NSString stringWithFormat:@"Section %ld, Cell %ld",
     (long)indexPath.section,
     (long)indexPath.row];

    result.accessoryType = UITableViewCellAccessoryDetailDisclosureButton;

  }

  return result;

}

- (NSInteger) tableView:(UITableView *)tableView
  numberOfRowsInSection:(NSInteger)section{
    return 10;
  }

- (void)viewDidLoad{
  [super viewDidLoad];

  self.myTableView = [[UITableView alloc]
                      initWithFrame:self.view.bounds
                      style:UITableViewStylePlain];

  [self.myTableView registerClass:[UITableViewCell class]
           forCellReuseIdentifier:MyCellIdentifier];

  self.myTableView.dataSource = self;

  self.myTableView.autoresizingMask =
    UIViewAutoresizingFlexibleWidth |
    UIViewAutoresizingFlexibleHeight;

  [self.view addSubview:self.myTableView];

}
```

## Обсуждение

Можно присваивать любые значения, определенные в перечне `UITableViewCellAccessoryType`, свойству `accessoryType` экземпляра класса `UITableViewCell`. Среди полезных дополнительных элементов следует особо отметить *индикатор подробного описания* и *кнопку детализации*[1]. Оба этих элемента содержат угловую скобку, подсказывающую пользователю, что если прикоснуться пальцем к соответствующей ячейке таблицы, то откроется новый вид или контроллер вида. Проще говоря, пользователь перейдет на новый экран с более подробной информацией об актуальном селекторе. Разница между двумя этими элементами заключается с том, что индикатор подробного описания не инициирует никакого события, а вот кнопка детализации при нажатии запускает событие, направляемое к делегату. Иными словами, эффект от нажатия кнопки не равен эффекту от нажатия самой ячейки. Следовательно, кнопка детализации позволяет пользователю осуществлять два разных, но связанных действия применительно к одной и той же строке.

На рис. 4.3 показаны два этих дополнительных элемента в табличном виде. В первой строке мы видим индикатор подробного описания, а во второй — кнопку детализации.

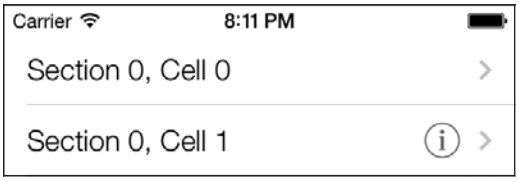


**Рис. 4.3.** Две ячейки табличного вида с различными дополнительными элементами

Если прикоснуться к любой кнопке детализации, присвоенной ячейке табличного вида, то сразу становится очевидно, что это, в сущности, самостоятельная кнопка. А теперь внимание — вопрос! Как табличный вид узнает, что пользователь нажал такую кнопку?

Как объяснялось ранее, табличный вид инициирует события, направляемые его объекту-делегату. Кнопка детализации из табличного вида также запускает событие, которое может быть принято объектом-делегатом табличного вида:

```
- (void)                          tableView:(UITableView *)tableView
 accessoryButtonTappedForRowWithIndexPath:(NSIndexPath *)indexPath{

  /* Делаем что-либо при нажатии дополнительной кнопки. */
  NSLog(@"Accessory button is tapped for cell at index path = %@",
        indexPath);

  UITableViewCell *ownerCell = [tableView cellForRowAtIndexPath:indexPath];
```

[1] Подробнее об этих элементах см.: http://habrahabr.ru/post/79280/. — *Примеч. пер.*

```
  NSLog(@"Cell Title = %@", ownerCell.textLabel.text);

}
```

Данный код ищет ячейку табличного вида, в которой была нажата кнопка детализации, и выводит в окне консоли содержимое текстовой метки данной ячейки. Напоминаю: чтобы отобразить окно консоли в Xcode, нужно выполнить команду Run ▸ Console (Запуск ▸ Консоль).

# 4.3. Создание специальных дополнительных элементов в ячейке табличного вида

## Постановка задачи

Дополнительных элементов, предоставляемых в iOS, недостаточно для решения задачи, и вы хотели бы создать собственные дополнительные элементы.

## Решение

Присвойте экземпляр класса `UIView` свойству `accessoryView` любого экземпляра класса `UITableViewCell`:

```
- (UITableViewCell *) tableView:(UITableView *)tableView
         cellForRowAtIndexPath:(NSIndexPath *)indexPath{

  UITableViewCell* cell = nil;

  cell = [tableView dequeueReusableCellWithIdentifier:MyCellIdentifier
                                         forIndexPath:indexPath];

  cell.textLabel.text = [NSString stringWithFormat:@"Section %ld, Cell %ld",
                         (long)indexPath.section,
                         (long)indexPath.row];

  UIButton *button = [UIButton buttonWithType:UIButtonTypeSystem];
  button.frame = CGRectMake(0.0f, 0.0f, 150.0f, 25.0f);

  [button setTitle:@"Expand"
         forState:UIControlStateNormal];

  [button addTarget:self
             action:@selector(performExpand:)
   forControlEvents:UIControlEventTouchUpInside];

  cell.accessoryView = button;

  return cell;

}
```

Как видите, в этом коде используется метод `performExpand:`. Он играет роль селектора для каждой кнопки. Вот определение данного метода:

```
- (void) performExpand:(id)paramSender{
  /* Обрабатываем событие нажатия кнопки */
}
```

В данном примере кода специальная создаваемая нами кнопка присваивается дополнительному виду в каждой строке выбранной таблицы. Результат показан на рис. 4.4.

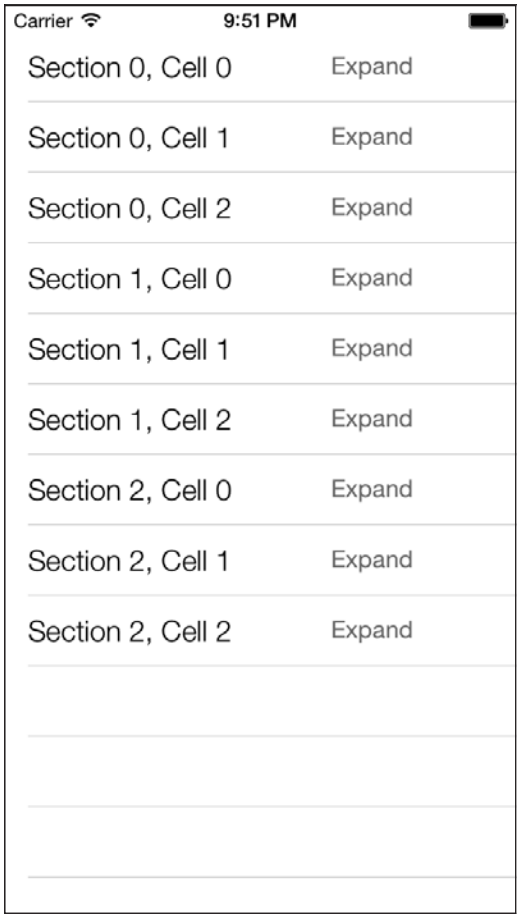


**Рис. 4.4.** Ячейки табличного вида со специальными дополнительными видами

## Обсуждение

Объект типа `UITableViewCell` содержит свойство `accessoryView`. Это тот вид, которому вы можете присвоить значение, если вас не вполне устраивают встроенные в SDK iOS дополнительные виды для табличных ячеек. После того как задано это

свойство, Cocoa Touch будет игнорировать значение свойства `accessoryType` и станет использовать вид, присвоенный свойству `accessoryView`, в качестве дополнительного элемента, который отображается в ячейке таблицы.

В коде, приведенном в подразделе «Решение» данного раздела, мы создаем кнопки для всех ячеек, находящихся в табличном виде. При нажатии кнопки в любой ячейке вызывается метод `performExpand:`. И если вы думаете примерно так же, как я, то вы уже стали задаваться вопросом: как же определить, к какой именно ячейке относится кнопка-отправитель? Итак, теперь нам нужно как-то связать кнопки с теми ячейками, к которым они относятся.

Один из способов разрешения этой ситуации связан с использованием свойства `tag` экземпляра кнопки. Это свойство-метка представляет собой обычное целое число, которое, как правило, используется для ассоциирования вида с другим объектом. Например, если вы хотите ассоциировать кнопку с третьей ячейкой в вашем табличном виде, то следует задать для свойства-метки этой кнопки значение `3`. Но здесь возникает проблема: в табличных видах есть разделы, и каждый раздел может содержать *n* ячеек. Следовательно, нам требуется возможность определить и раздел таблицы, и ячейку, которая владеет нашей кнопкой. А поскольку значением свойства-метки может быть только одно целое число, эта задача существенно усложняется. Поэтому мы можем отказаться от метки и вместо работы с ней запрашивать вышестоящий вид о дополнительном виде, рекурсивно проходя вверх по цепочке видов, пока не найдем ячейку типа `UITableViewCell`, вот так:

```
- (UIView *) superviewOfType:(Class)paramSuperviewClass
  forView:(UIView *)paramView{

    if (paramView.superview != nil){
      if ([paramView.superview isKindOfClass:paramSuperviewClass]){
        return paramView.superview;
      } else {
        return [self superviewOfType:paramSuperviewClass
        forView:paramView.superview];
      }

    }

    return nil;

  }

- (void) performExpand:(UIButton *)paramSender{

  /* Обрабатываем событие нажатия кнопки */
  __unused UITableViewCell *parentCell =
    (UITableViewCell *)[self superviewOfType:[UITableViewCell class]
                                     forView:paramSender];

  /* Теперь, если желаете, можете еще что-нибудь сделать с ячейкой */

}
```

Здесь мы используем простой рекурсивный метод, принимающий вид (в данном случае нашу кнопку) и имя класса (в данном случае `UITableViewCell`), а затем просматриваем иерархию вида, являющегося вышестоящим для данного, чтобы найти вышестоящий вид, относящийся к интересующему нас классу. Итак, он начинает работу с вида, являющегося вышестоящим для заданного, и если этот вышестоящий вид не относится к требуемому типу, то просматривает и его вышестоящий вид, и так до тех пор, пока не найдет один из вышестоящих видов, относящийся к требуемому классу. Как видите, в качестве первого параметра метода `superviewOfType:forView:` мы используем структуру `Class`. В этом типе данных может содержаться имя любого класса из языка Objective-C, и это весьма кстати, если вы ищете или запрашиваете у программиста конкретные имена классов.

# 4.4. Обеспечение удаления смахиванием в ячейках табличных видов

## Постановка задачи

Необходимо предоставить пользователям приложения возможность без труда удалять строки из табличного вида.

## Решение

Реализуйте в делегате табличного вида селектор `tableView:editingStyleForRowAtIndexPath:`, а в источнике данных табличного вида — селектор `tableView:commitEditingStyle:forRowAtIndexPath:`:

```
- (UITableViewCellEditingStyle)tableView:(UITableView *)tableView
           editingStyleForRowAtIndexPath:(NSIndexPath *)indexPath{

    return UITableViewCellEditingStyleDelete;

}

- (void) setEditing:(BOOL)editing
           animated:(BOOL)animated{

    [super setEditing:editing
             animated:animated];

    [self.myTableView setEditing:editing
                        animated:animated];

}

- (void)  tableView:(UITableView *)tableView
 commitEditingStyle:(UITableViewCellEditingStyle)editingStyle
```

```
  forRowAtIndexPath:(NSIndexPath *)indexPath{

    if (editingStyle == UITableViewCellEditingStyleDelete){

        /* Сначала удаляем этот объект из источника данных */
        [self.allRows removeObjectAtIndex:indexPath.row];

        /* Потом удаляем ассоциированную с ним ячейку из табличного вида */
        [tableView deleteRowsAtIndexPaths:@[indexPath]
                         withRowAnimation:UITableViewRowAnimationLeft];

    }

}
```

Метод `tableView:editingStyleForRowAtIndexPath:` позволяет выполнять операции удаления. Он вызывается табличным видом, а его возвращаемое значение определяет, какие операции пользователь может делать в табличном виде (вставлять информацию, удалять информацию и т. д.). Метод `tableView:commitEditingStyle:forRowAtIndexPath:` выполняет затребованную пользователем операцию удаления. Второй из указанных методов определяется в делегате, но его функционал несколько перегружен: этот метод применяется не только для удаления данных, но и для удаления строк из таблицы.

## Обсуждение

Табличный вид реагирует на жест смахивания (Swipe), отображая кнопку в правой части затронутой строки (рис. 4.5). Как видите, табличный вид *не находится* в режиме редактирования, но эта кнопка позволяет пользователю удалить строку.

Такой режим активизируется путем реализации метода `tableView:editingStyleForRowAtIndexPath:` (определяемого в протоколе `UITableViewDelegate`), чье возвращаемое значение указывает, будут ли в таблице разрешаться операции вставки, или удаления, или обе эти операции, или ни одна из них. Реализуя метод `tableView:commitEditingStyle:forRowAtIndexPath:` в источнике данных табличного вида, можно также получать уведомление о том, какую операцию выполнил пользователь, вставку или удаление.

Второй параметр метода `deleteRowsAtIndexPaths:withRowAnimation:` позволяет указывать метод анимации, который будет выполняться при удалении строк из табличного вида. В примере мы задали, что удаляемые строки будут уходить с экрана в направлении справа налево.

# 4.5. Создание верхних и нижних колонтитулов в табличных видах

## Постановка задачи

Необходимо создать в таблице верхний и/или нижний колонтитул.

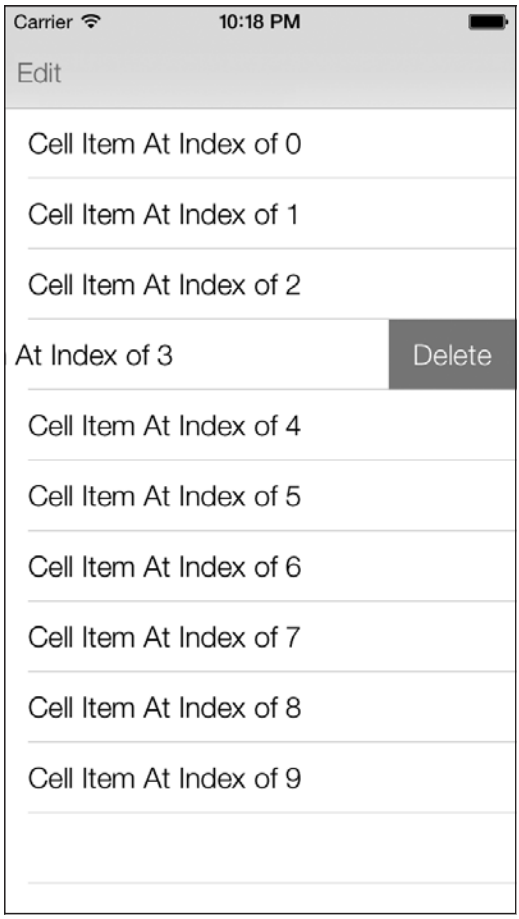


**Рис. 4.5.** Кнопка для удаления, появляющаяся в ячейке табличного вида

## Решение

Создайте вид (это может быть подпись, вид с изображением или любой другой класс, прямо или опосредованно производимый от `UIView`) и присвойте этот вид верхнему и/или нижнему колонтитулу табличного раздела. Кроме того, как вы вскоре увидите, для верхнего или нижнего колонтитулов можно выделять конкретное количество точек.

## Обсуждение

Табличный вид может иметь несколько верхних и нижних колонтитулов. У каждого раздела табличного вида может быть свой верхний и нижний колонтитул, так что если у вас в табличном виде три раздела, то в нем может быть максимум три верхних и три нижних колонтитула. Вы *не обязаны* создавать верхние и нижние

колонтитулы в каком-либо из разделов и сами решаете, сообщать или нет табличному виду, что в определенном его разделе будут верхний и нижний колонтитулы. Эти виды-колонтитулы передаются табличному виду через его делегат — если вы решите их сделать. Верхние и нижние колонтитулы становятся частью табличного вида. Это означает, что, когда содержимое таблицы прокручивается, одновременно с ним прокручиваются и колонтитулы табличных разделов. Рассмотрим примеры верхнего и нижнего колонтитулов в табличном виде (рис. 4.6).

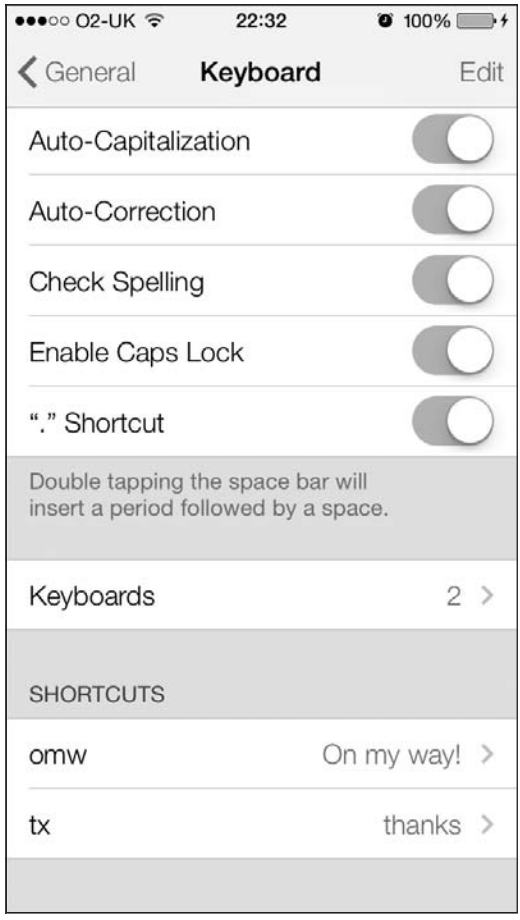


**Рис. 4.6.** Нижний колонтитул в верхнем разделе и верхний колонтитул Shortcuts (Быстрый доступ) в последнем разделе табличного вида

Как видите, в верхнем разделе (там, где находятся элементы Check Spelling (Проверка правописания) и Enable Caps Lock (Зафиксировать верхний регистр)) в нижнем колонтитуле написано: Double tapping the space bar will insert a period followed by a space (Двойное нажатие клавиши пробела вставляет точку, за которой следует пробел). Это нижний колонтитул верхнего раздела рассматриваемого вида. Причина, по которой этот фрагмент находится именно в нижнем, а не в верхнем колонтитуле, в том,

что он прикреплен к нижней, а не к верхней части раздела. В последнем разделе данной таблицы также есть верхний колонтитул, на котором написано Shortcuts (Быстрый доступ). Здесь, наоборот, колонтитул является верхним, а не нижним, так как он прикреплен к верхней части раздела.

Для указания высоты верхнего и нижнего колонтитулов в разделе табличного вида применяются методы, определяемые в протоколе UITableViewDataSource. Чтобы задать сам вид, который будет соответствовать верхнему/нижнему колонтитулу в разделе табличного вида, нужно использовать методы, определяемые в протоколе UITableViewDelegate.

Идем дальше. Создадим простое приложение, внутри которого будет табличный вид. Потом сделаем две метки типа `UILabel`, одна будет играть роль верхнего колонтитула, а другая — нижнего в единственном разделе нашего табличного вида. Этот раздел будет заполнен всего тремя ячейками. В верхнем колонтитуле мы напишем Section 1 Header (Верхний колонтитул раздела 1), а в нижнем — Section 1 Footer (Нижний колонтитул раздела 1). Начнем с файла реализации контроллера вида, где определим табличный вид:

```
#import "ViewController.h"

static NSString *CellIdentifier = @"CellIdentifier";

@interface ViewController () <UITableViewDelegate, UITableViewDataSource>
@property (nonatomic, strong) UITableView *myTableView;
@end

@implementation ViewController
```

После этого создадим сгруппированный табличный вид и загрузим в него три ячейки:

```
- (UITableViewCell *) tableView:(UITableView *)tableView
         cellForRowAtIndexPath:(NSIndexPath *)indexPath{

  UITableViewCell *cell = nil;

  cell = [tableView dequeueReusableCellWithIdentifier:CellIdentifier
                                         forIndexPath:indexPath];

  cell.textLabel.text = [[NSString alloc] initWithFormat:@"Cell %ld",
                         (long)indexPath.row];

  return cell;

}

- (NSInteger) tableView:(UITableView *)tableView
  numberOfRowsInSection:(NSInteger)section{
  return 3;
}
```

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

  self.myTableView =
  [[UITableView alloc] initWithFrame:self.view.bounds
                               style:UITableViewStyleGrouped];
  [self.myTableView registerClass:[UITableViewCell class]
           forCellReuseIdentifier:CellIdentifier];


  self.myTableView.dataSource = self;
  self.myTableView.delegate = self;
  self.myTableView.autoresizingMask = UIViewAutoresizingFlexibleWidth |
  UIViewAutoresizingFlexibleHeight;

  [self.view addSubview:self.myTableView];

}
```

И тут начинается самое интересное. Мы можем воспользоваться двумя важными методами (определяемыми в протоколе `UITableViewDelegate`), чтобы сделать метку и для верхнего и для нижнего колонтитула того раздела, который мы загрузили в табличный вид. Вот эти методы:

- `tableView:viewForHeaderInSection:` — ожидает возвращаемого значения типа `UIView`. Вид, возвращаемый этим методом, отобразится как верхний колонтитул раздела и будет указан в параметре `viewForHeaderInSection`;
- `tableView:viewForFooterInSection:` — ожидает возвращаемого значения типа `UIView`. Вид, возвращаемый этим методом, отобразится как нижний колонтитул раздела и будет указан в параметре `viewForFooterInSection`.

Теперь наша задача заключается в том, чтобы реализовать эти методы и вернуть экземпляр `UILabel`. На метке верхнего колонтитула мы укажем текст Section 1 Header (Верхний колонтитул раздела 1), а на метке нижнего — Section 1 Footer (Нижний колонтитул раздела 1), как и планировали:

```
- (UILabel *) newLabelWithTitle:(NSString *)paramTitle{
  UILabel *label = [[UILabel alloc] initWithFrame:CGRectZero];
  label.text = paramTitle;
  label.backgroundColor = [UIColor clearColor];
  [label sizeToFit];
  return label;
}

- (UIView *)   tableView:(UITableView *)tableView
  viewForHeaderInSection:(NSInteger)section{

  if (section == 0){
    return [self newLabelWithTitle:@"Section 1 Header"];
  }
```

```
  return nil;

}

- (UIView *)   tableView:(UITableView *)tableView
  viewForFooterInSection:(NSInteger)section{

  if (section == 0){
    return [self newLabelWithTitle:@"Section 1 Footer"];
  }

  return nil;

}
```

Если теперь запустить приложение в эмуляторе, получится такая картинка, как на рис. 4.7.

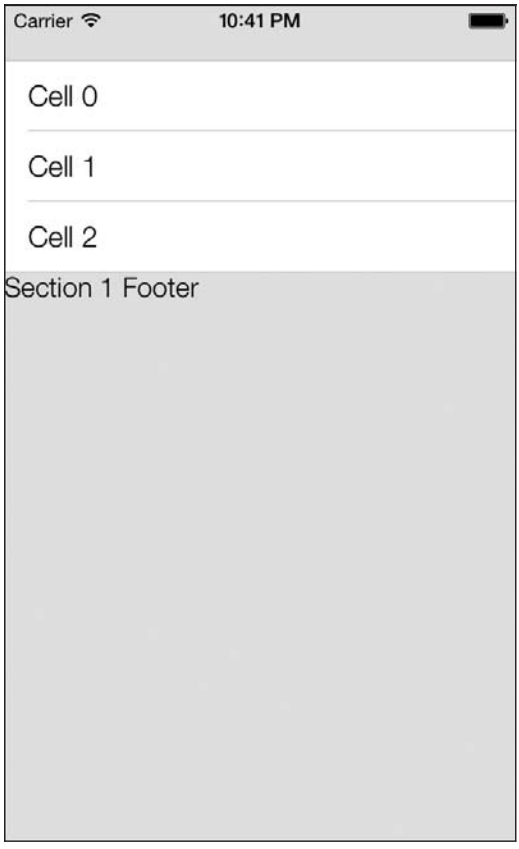


**Рис. 4.7.** Метки для верхнего и нижнего колонтитулов табличного вида, выровненные неправильно

Причина такого неправильного выравнивания в том, что родительский вид не знает высоты видов-меток. Для указания высоты видов верхнего и нижнего колонтитулов следует использовать два следующих метода, определяемых в протоколе `UITableViewDelegate`:

- `tableView:heightForHeaderInSection:` — возвращаемое значение данного метода относится к типу `CGFloat`. Оно указывает высоту верхнего колонтитула раздела табличного вида. Индекс раздела передается в параметре `heightForHeaderInSection`;
- `tableView:heightForFooterInSection:` — возвращаемое значение данного метода относится к типу `CGFloat`. Оно указывает высоту нижнего колонтитула раздела табличного вида. Индекс раздела передается в параметре `heightForHeaderInSection`.

```
- (CGFloat) tableView:(UITableView *)tableView
  heightForHeaderInSection:(NSInteger)section{

    if (section == 0){
      return 30.0f;
    }
    return 0.0f;
}

- (CGFloat) tableView:(UITableView *)tableView
  heightForFooterInSection:(NSInteger)section{

    if (section == 0){
      return 30.0f;
    }

  return 0.0f;

}
```

Запустив это приложение, вы увидите, что теперь метки верхнего и нижнего колонтитулов имеют фиксированную высоту. Но в написанном нами коде все еще остается какая-то ошибка — дело в левом поле меток верхнего и нижнего колонтитулов. В этом можно убедиться, взглянув на рис. 4.8.

Причина заключается в том, что по умолчанию табличный вид размещает верхний и нижний колонтитулы в точке с координатой `0.0f` по оси *X*. Можно подумать, что эта проблема решается изменением контуров меток верхнего и нижнего колонтитулов, но, к сожалению, это мнение ошибочно. Проблема решается созданием универсального вида `UIView`, где и размещаются метки для верхнего и нижнего колонтитулов. Возвратите в качестве верхнего/нижнего колонтитула такой универсальный вид, но измените положение меток по оси *X* в этом виде.

Теперь изменим реализацию методов `tableView:viewForHeaderInSection:` и `tableView:viewForFooterInSection:`:

```
- (UIView *)  tableView:(UITableView *)tableView
  viewForHeaderInSection:(NSInteger)section{
```

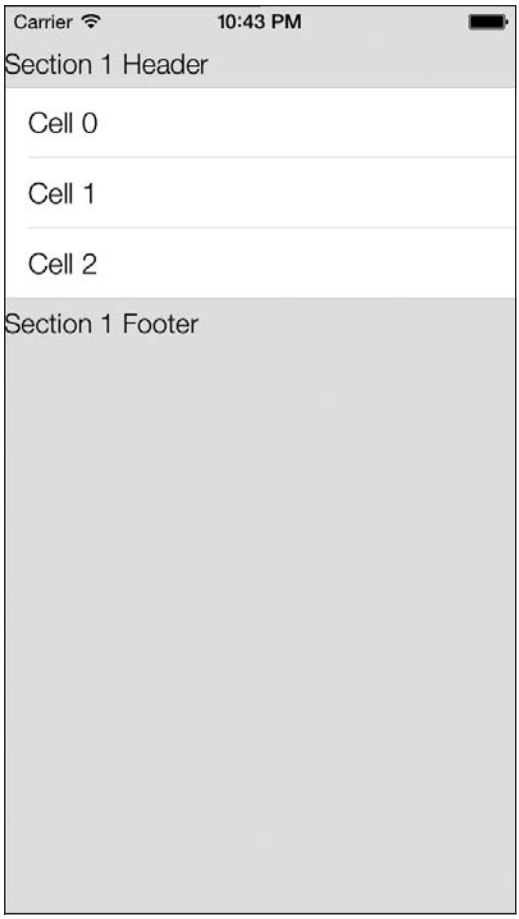


**Рис. 4.8.** Левые поля меток в верхнем и нижнем колонтитулах — неправильные

```
UIView *header = nil;
if (section == 0){
    UILabel *label = [self newLabelWithTitle:@"Section 1 Header"];
/* Перемещаем метку на 10 точек вправо. */
label.frame = CGRectMake(label.frame.origin.x + 10.0f,
                         5.0f, /* Опускаемся на 5 точек вниз
                                  по оси y. */
                         label.frame.size.width,
                         label.frame.size.height);

/* Делаем ширину содержащего вида на 10 точек больше,
   чем ширина метки, так как для метки требуется
   10 дополнительных точек ширины в левом поле. */

CGRect resultFrame = CGRectMake(0.0f,
                                0.0f,
```

```
                                    label.frame.size.width + 10.0f,
                                    label.frame.size.height);
    header = [[UIView alloc] initWithFrame:resultFrame];
    [header addSubview:label];

  }

  return header;

}

- (UIView *)  tableView:(UITableView *)tableView
 viewForFooterInSection:(NSInteger)section{

  UIView *footer = nil;
  if (section == 0){

    UILabel *label = [[UILabel alloc] initWithFrame:CGRectZero];

    /* Перемещаем метку на 10 точек вправо. */
    label.frame = CGRectMake(label.frame.origin.x + 10.0f,
                             5.0f, /* Опускаемся на 5 точек вниз по оси y*/
                             label.frame.size.width,
                             label.frame.size.height);

    /* Делаем ширину содержащего вида на 10 точек больше,
       чем ширина метки, так как для метки требуется
       10 дополнительных точек ширины в левом поле. */
    CGRect resultFrame = CGRectMake(0.0f,
                                    0.0f,
                                    label.frame.size.width + 10.0f,
                                    label.frame.size.height);
    footer = [[UIView alloc] initWithFrame:resultFrame];
    [footer addSubview:label];

  }

  return footer;

}
```

Теперь, запустив приложение, вы получите примерно такой результат, как на рис. 4.9.

Пользуясь изученными здесь методами, вы также можете размещать изображения в верхнем и нижнем колонтитулах табличных видов. Экземпляры класса `UIImageView` являются производными от класса `UIView`, поэтому вы легко можете ставить картинки в виды для изображений и возвращать их как верхние/нижние колонтитулы табличного вида. Если вы не собираетесь помещать в верхних и нижних колонтитулах табличных видов ничего, кроме текста, то можете пользоваться двумя удобными методами, определяемыми в протоколе `UITableViewDataSource`. Эти

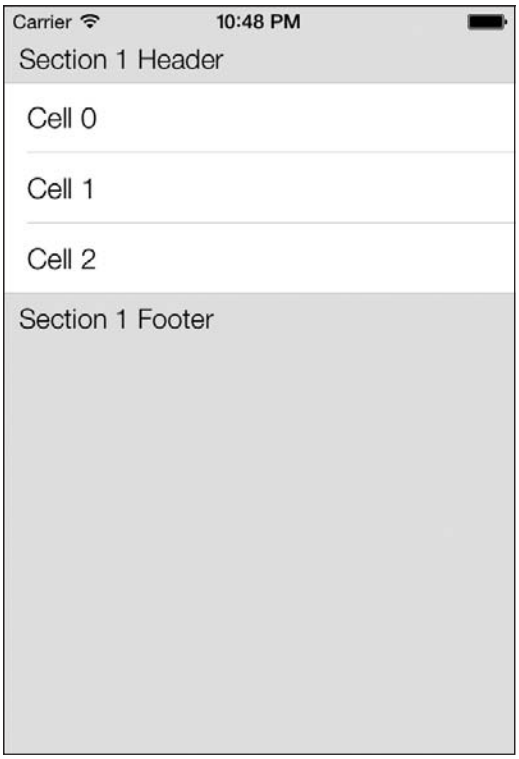


**Рис. 4.9.** В табличном виде отображаются метки верхнего и нижнего колонтитулов

методы избавят вас от массы проблем. Чтобы не создавать собственные метки и не возвращать их как верхние/нижние колонтитулы табличного вида, просто пользуйтесь следующими методами:

- `tableView:titleForHeaderInSection:` — возвращаемое значение этого метода относится к типу `NSString`. Табличный вид будет автоматически помещать в метке строку, которая будет отображаться как верхний колонтитул раздела, указываемый в параметре `titleForHeaderInSection`;
- `tableView:titleForFooterInSection:` — возвращаемое значение этого метода относится к типу `NSString`. Табличный вид будет автоматически помещать в метке строку, которая будет отображаться как нижний колонтитул раздела, указываемый в параметре `titleForFooterInSection`.

Итак, чтобы упростить код приложения, избавимся от реализаций методов `tableView:viewForHeaderInSection:` и `tableView:viewForFooterInSection:`, заменив их реализациями методов `tableView:titleForHeaderInSection:` и `tableView:titleForFooterInSection:`:

```
- (NSString *) tableView:(UITableView *)tableView
  titleForHeaderInSection:(NSInteger)section{

    if (section == 0){
```

```
      return @"Section 1 Header";
    }

  return nil;

}
- (NSString *) tableView:(UITableView *)tableView
  titleForFooterInSection:(NSInteger)section{

    if (section == 0){
      return @"Section 1 Footer";
    }

  return nil;

}
```

Теперь запустите ваше приложение в эмуляторе iPhone. Вы увидите, что табличный вид автоматически создал для верхнего колонтитула метку, выровненную по левому краю, а для нижнего колонтитула — метку, выровненную по центру, и поместил их в единственном разделе табличного вида. В iOS 7 по умолчанию верхний и нижний колонтитулы выравниваются по левому краю. В более ранних версиях iOS верхний колонтитул выравнивался по левому краю, а нижний — по центру. В любой версии выравнивание этих меток может задаваться табличным видом (рис. 4.10).

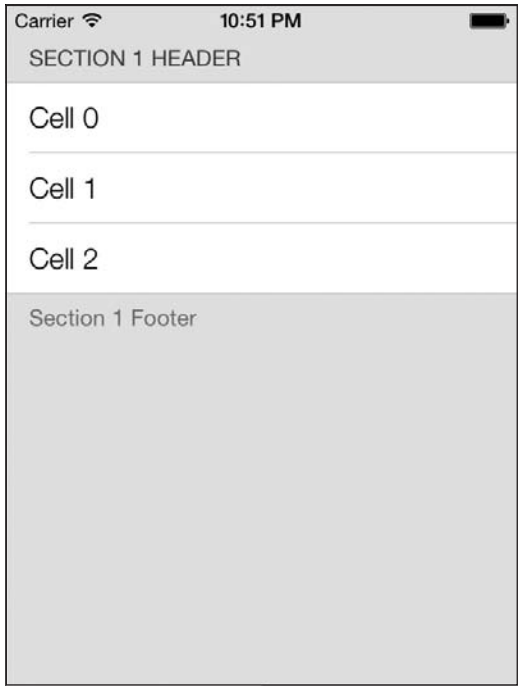


**Рис. 4.10.** Табличный вид, в верхнем и нижнем колонтитулах которого отображается текст

# 4.6. Отображение контекстных меню в ячейках табличных видов

## Постановка задачи

Необходимо дать пользователям возможность применять операции копирования и вставки. Предполагается, что при этом пользователь будет удерживать пальцем определенную ячейку таблицы на экране устройства, где отображается приложение.

## Решение

Реализуйте следующие три метода протокола `UITableViewDelegate` в объекте-делегате вашего табличного вида.

- `tableView:shouldShowMenuForRowAtIndexPath:` — возвращаемое значение данного вида относится к типу `BOOL`. Если вернуть от этого метода значение `YES`, то система iOS отобразит для ячейки табличного вида контекстное меню. Индекс этой ячейки будет передан вам в параметре `shouldShowMenuForRowAtIndexPath`.
- `tableView:canPerformAction:forRowAtIndexPath:withSender:` — возвращаемое значение данного метода также относится к типу `BOOL`. Как только вы позволите iOS отображать контекстное меню для ячейки табличного вида, iOS вызовет этот метод несколько раз и сообщит вам селектор действия. После этого вы сможете решить, следует ли отображать это действие в командах контекстного меню. Итак, если iOS спрашивает вас, хотите ли вы отобразить для пользователя меню Copy (Копировать), то рассматриваемый метод будет вызван в объекте-делегате вашего табличного вида и параметр `canPerformAction` данного метода будет равен `@selector(copy:)`. Подробнее этот вопрос рассматривается в подразделе «Обсуждение» данного раздела.
- `tableView:performAction:forRowAtIndexPath:withSender:` — как только вы разрешите отобразить определенное действие в списке вариантов контекстного меню ячейки табличного вида, возникает такая ситуация: когда пользователь выбирает это действие в меню, данный метод вызывается в объекте-делегате вашего табличного вида. Здесь нужно сделать все необходимое, чтобы удовлетворить пользовательский запрос. Например, если пользователь выбрал меню Copy (Копировать), то вы должны применить буфер обмена (Pasteboard), куда помещается содержимое из ячейки табличного вида.

## Обсуждение

Табличный вид может дать системе iOS ответ «да» или «нет», позволив или не позволив отобразить доступные системные элементы меню для данной табличной ячейки. iOS пытается вывести контекстное меню для табличной ячейки, когда пользователь удерживает эту ячейку пальцем в течение определенного временного промежутка — обычно примерно 1 с. Затем iOS пытается узнать табличный вид,

одна из ячеек которого инициировала появление контекстного меню на экране. Если табличный вид ответит, то iOS сообщит ему, какие команды можно отобразить в контекстном меню, а табличный вид сможет утвердительно или отрицательно отреагировать на каждый из этих вариантов. Например, если доступны пять вариантов (элементов) и табличный вид отвечает «да» на два из них, то будут отображены только два этих элемента.

После того как элементы меню будут показаны пользователю, последний может нажать либо на любой из этих элементов, либо на экран за пределами контекстного меню, чтобы убрать это меню. Как только пользователь прикоснется к одному из элементов меню, iOS пошлет табличному виду сообщение от делегата, информирующее табличный вид о том, какой именно элемент меню был выбран пользователем. В зависимости от полученной информации табличный вид может решить, что делать с выбранным действием.

Предлагаю сначала рассмотреть, какие действия доступны в контекстном меню ячейки табличного вида. Поэтому создадим табличный вид и отобразим в нем несколько ячеек:

```
- (NSInteger) tableView:(UITableView *)tableView
  numberOfRowsInSection:(NSInteger)section{
  return 3;
}

- (UITableViewCell *) tableView:(UITableView *)tableView
         cellForRowAtIndexPath:(NSIndexPath *)indexPath{

  UITableViewCell *cell = nil;


cell = [tableView dequeueReusableCellWithIdentifier:CellIdentifier
                                       forIndexPath:indexPath];

cell.textLabel.text = [[NSString alloc]
                       initWithFormat:@"Section %ld Cell %ld",
                       (long)indexPath.section,
                       (long)indexPath.row];

  return cell;

}

- (void)viewDidLoad{
  [super viewDidLoad];

  self.myTableView = [[UITableView alloc]
                      initWithFrame:self.view.bounds
                      style:UITableViewStylePlain];
  [self.myTableView registerClass:[UITableViewCell class]
           forCellReuseIdentifier:CellIdentifier];
```

```
  self.myTableView.autoresizingMask = UIViewAutoresizingFlexibleWidth |
                                      UIViewAutoresizingFlexibleHeight;

  self.myTableView.dataSource = self;
  self.myTableView.delegate = self;

  [self.view addSubview:self.myTableView];

}
```

Теперь реализуем три упомянутых ранее метода, определенных в протоколе `UITableViewDelegate`, и просто преобразуем доступные действия (типа `SEL`) в строку, после чего выведем доступные результаты на консоль:

```
- (BOOL)                tableView:(UITableView *)tableView
  shouldShowMenuForRowAtIndexPath:(NSIndexPath *)indexPath{

  /* Разрешаем отображение контекстного меню для каждой ячейки */
  return YES;

}

- (BOOL) tableView:(UITableView *)tableView
  canPerformAction:(SEL)action
 forRowAtIndexPath:(NSIndexPath *)indexPath
        withSender:(id)sender{

  NSLog(@"%@", NSStringFromSelector(action));

  /* Пока разрешим любые действия. */
  return YES;
}

- (void) tableView:(UITableView *)tableView
     performAction:(SEL)action
 forRowAtIndexPath:(NSIndexPath *)indexPath
        withSender:(id)sender{

  /* Пока оставим пустым. */

}
```

А теперь запустим приложение в эмуляторе или на устройстве. После этого мы увидим, что в табличный вид загружены три ячейки. Удерживайте на ячейке палец (если работаете с устройством) или указатель мыши (если с эмулятором) и смотрите, какая информация появляется в окне консоли:

```
cut:
copy:
select:
selectAll:
paste:
```

```
delete:
_promptForReplace:
_showTextStyleOptions:
_define:
_addShortcut:
_accessibilitySpeak:
_accessibilitySpeakLanguageSelection:
_accessibilityPauseSpeaking:
makeTextWritingDirectionRightToLeft:
makeTextWritingDirectionLeftToRight:
```

Все это действия, которые система iOS позволяет вывести на экран для пользователя, если такие действия вам понадобятся. Допустим, вы хотите разрешить пользователям операцию копирования (Copy). Для этого перед отображением команды просто найдите в методе `tableView:canPerformAction:forRowAtIndexPath:withSender:`, на какое действие запрашивает у вас разрешение система iOS, а потом верните значение `YES` или `NO`:

```
- (BOOL) tableView:(UITableView *)tableView
  canPerformAction:(SEL)action
 forRowAtIndexPath:(NSIndexPath *)indexPath
        withSender:(id)sender{

  if (action == @selector(copy:)){
    return YES;
  }

  return NO;
}
```

На следующем этапе перехватываем информацию о том, какой именно элемент был выбран пользователем в контекстном меню. В зависимости от того, что выяснится, мы можем совершить нужное действие. Например, если пользователь выберет в контекстном меню команду Copy (Копировать) (рис. 4.11), мы воспользуемся `UIPasteBoard`, чтобы скопировать эту ячейку в компоновочный буфер и иметь возможность применять ее позже:

```
- (void) tableView:(UITableView *)tableView
     performAction:(SEL)action
 forRowAtIndexPath:(NSIndexPath *)indexPath
        withSender:(id)sender{

  if (action == @selector(copy:)){

    UITableViewCell *cell = [tableView cellForRowAtIndexPath:indexPath];
    UIPasteboard *pasteBoard = [UIPasteboard generalPasteboard];
    [pasteBoard setString:cell.textLabel.text];

  }

}
```

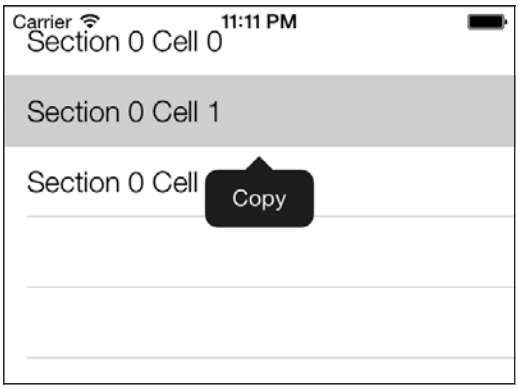


**Рис. 4.11.** Команда Copy (Копировать), отображенная в контекстном меню ячейки табличного вида

# 4.7. Перемещение ячеек и разделов в табличных видах

## Постановка задачи

Требуется перемещать и тасовать ячейки и разделы внутри табличного вида, сопровождая весь процесс плавной и интуитивно понятной анимацией.

## Решение

Используйте метод `moveSection:toSection:` табличного вида, чтобы переместить раздел на новое место. Кроме того, можно применять метод `moveRowAtIndexPath:toIndexPath:`, чтобы перемещать ячейку табличного вида на новое место с того места, которое она сейчас занимает.

## Обсуждение

Процесс перемещения разделов и ячеек таблицы отличается от их замены. Рассмотрим пример, помогающий лучше понять эту разницу. Допустим, у нас есть табличный вид с тремя разделами, A, B и C. Если передвинуть раздел A к разделу C, то табличный вид заметит это и переместит раздел B туда, где до этого находился раздел A. Но если раздел B будет перемещен на место раздела C, то табличному виду вообще не придется перемещать раздел A, так как он находится «выше» двух перемещаемых разделов и не участвует в передвижениях B и C. В данном случае раздел B попадет на место раздела C, а раздел C — на место раздела B. Такая же логика применяется в табличных видах при перемещении ячеек.

Для демонстрации таких взаимодействий создадим табличный вид и загрузим в него три раздела, в каждом из которых есть три собственные ячейки. Начнем с файла реализации контроллера вида:

```
#import "ViewController.h"

static NSString *CellIdentifier = @"CellIdentifier";

@interface ViewController () <UITableViewDelegate, UITableViewDataSource>
@property (nonatomic, strong) UITableView *myTableView;
@property (nonatomic, strong) NSMutableArray *arrayOfSections;
@end
```

Контроллер вида становится источником данных для табличного вида. В табличном виде есть разделы, а в каждом разделе — ячейки. Мы, в сущности, работаем с массивом массивов: массив первого порядка содержит разделы, а каждый раздел, в свою очередь, является массивом, содержащим ячейки. Отвечать за этот функционал будет элемент `arrayOfSections`, определяемый в заголовочном файле контроллера вида. Итак, заполним этот массив:

```
- (NSMutableArray *) newSectionWithIndex:(NSUInteger)paramIndex
                              withCellCount:(NSUInteger)paramCellCount{

  NSMutableArray *result = [[NSMutableArray alloc] init];

  NSUInteger counter = 0;
  for (counter = 0;
       counter < paramCellCount;
       counter++){

    [result addObject:[[NSString alloc] initWithFormat:@"Section %lu
                       Cell %lu",
                       (unsigned long)paramIndex,
                       (unsigned long)counter+1]];

  }

  return result;

}

- (NSMutableArray *) arrayOfSections{
  if (_arrayOfSections == nil){
    NSMutableArray *section1 = [self newSectionWithIndex:1
                                                cellCount:3];
    NSMutableArray *section2 = [self newSectionWithIndex:2
                                                cellCount:3];
    NSMutableArray *section3 = [self newSectionWithIndex:3
                                                cellCount:3];
  _arrayOfSections = [[NSMutableArray alloc] initWithArray:@[
                                                           section1,
                                                           section2,
                                                           section3
                                                           ]
```

```
                    ];
  }
  return _arrayOfSections;
}
```

Затем мы инстанцируем табличный вид и реализуем необходимые методы в протоколе `UITableViewDataSource`, чтобы заполнить табличный вид данными:

```
- (NSInteger) numberOfSectionsInTableView:(UITableView *)tableView{

  return self.arrayOfSections.count;

}

- (NSInteger) tableView:(UITableView *)tableView
  numberOfRowsInSection:(NSInteger)section{

    NSMutableArray *sectionArray = self.arrayOfSections[section];
    return sectionArray.count;

  }

- (UITableViewCell *) tableView:(UITableView *)tableView
         cellForRowAtIndexPath:(NSIndexPath *)indexPath{

  UITableViewCell *cell = nil;

  cell = [tableView dequeueReusableCellWithIdentifier:CellIdentifier
                                         forIndexPath:indexPath];

  NSMutableArray *sectionArray = self.arrayOfSections[indexPath.section];

  cell.textLabel.text = sectionArray[indexPath.row];

  return cell;

}

- (void)viewDidLoad{
  [super viewDidLoad];

  self.myTableView =
  [[UITableView alloc] initWithFrame:self.view.bounds
                               style:UITableViewStyleGrouped];

  [self.myTableView registerClass:[UITableViewCell class]
           forCellReuseIdentifier:CellIdentifier];

  self.myTableView.autoresizingMask =
    UIViewAutoresizingFlexibleWidth |
    UIViewAutoresizingFlexibleHeight;
```

```
  self.myTableView.delegate = self;
  self.myTableView.dataSource = self;

  [self.view addSubview:self.myTableView];

}
```

Теперь посмотрим, что получается. Сначала проверим, как разделы перемещаются на новое место. Напишем метод, который будет перемещать раздел 1 на место раздела 3:

```
- (void) moveSection1ToSection3{

  NSMutableArray *section1 = [self.arrayOfSections objectAtIndex:0];
  [self.arrayOfSections removeObject:section1];
  [self.arrayOfSections addObject:section1];

  [self.myTableView moveSection:0
                      toSection:2];

}
```

Оставляю на ваш выбор окончательное решение о том, как инициировать этот метод, ведь на данный момент у нас в пользовательском интерфейсе нет специальной кнопки для этой цели. Можно просто создать навигационный контроллер и разместить на нем навигационную кнопку, которая и будет запускать данный метод.

Как только вы запустите приложение в обычном режиме, на экране появятся разделы с 1-го по 3-й (рис. 4.12).

После запуска метода `moveSection1ToSection3` вы увидите, что раздел 1 переходит на место раздела 3, раздел 3 переходит на место, ранее занятое разделом 2, и, наконец, раздел 2 перемещается на то место, где раньше находился раздел 1 (рис. 4.13).

Перемещение ячеек очень напоминает перемещение разделов. Для этого нужно просто пользоваться методом `moveRowAtIndexPath:toIndexPath:`. Не забывайте, что ячейка может перемещаться либо в пределах одного раздела, либо из одного раздела в другой. Начнем с простого — переместим ячейку 1 из 1-го раздела на место ячейки 2 того же раздела и посмотрим, что получится:

```
- (void) moveCell1InSection1ToCell2InSection1{

  NSMutableArray *section1 = [self.arrayOfSections objectAtIndex:0];
  NSString *cell1InSection1 = [section1 objectAtIndex:0];
  [section1 removeObject:cell1InSection1];
  [section1 insertObject:cell1InSection1
                 atIndex:1];

  NSIndexPath *sourceIndexPath = [NSIndexPath indexPathForRow:0
                                                    inSection:0];
  NSIndexPath *destinationIndexPath = [NSIndexPath indexPathForRow:1
                                                         inSection:0];
```

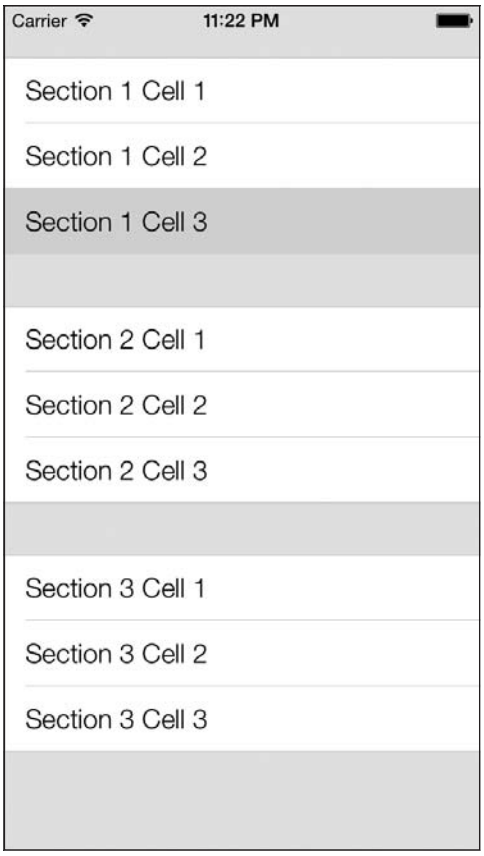


**Рис. 4.12.** Табличный вид с тремя разделами, в каждом из которых находятся по три ячейки

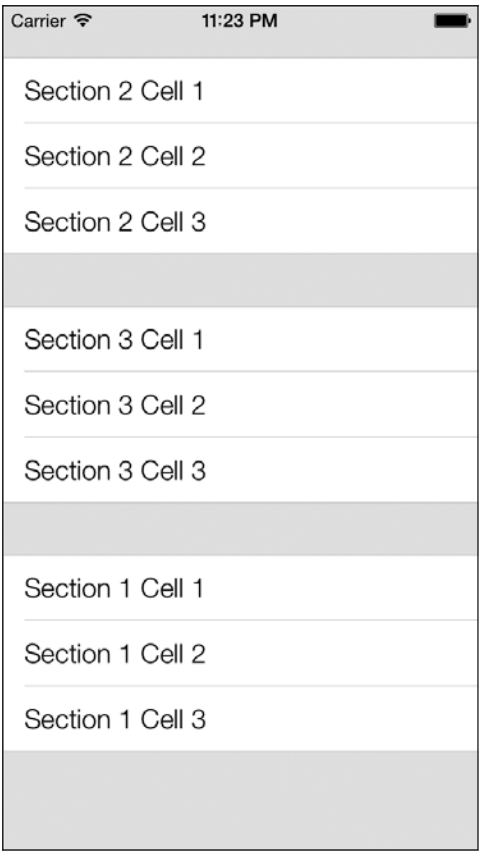


**Рис. 4.13.** Раздел 1 перешел на место раздела 3, после чего последовательно переместились и другие разделы

```
 [self.myTableView moveRowAtIndexPath:sourceIndexPath
                          toIndexPath:destinationIndexPath];

}
```

Что же происходит в этом коде? Нам нужно гарантировать, что в источнике данных содержится корректная информация, которая отобразится в табличном виде по окончании всех перестановок. Поэтому сначала убираем ячейку 1 в разделе 1. В результате ячейка 2 переходит на место, освобожденное ячейкой 1, а ячейка 3 — на место, ранее занятое ячейкой 2. В массиве остается всего 2 ячейки. Потом мы вставляем ячейку 1 в индекс 1 (второй объект) массива. Таким образом, в массиве будут содержаться ячейка 2, ячейка 1, а потом ячейка 3. И вот теперь мы на самом деле переместили ячейки в табличном виде.

Теперь немного усложним задачу. Попробуем переместить ячейку 2 из раздела 1 на место ячейки 1 из раздела 2:

```
- (void) moveCell2InSection1ToCell1InSection2{

  NSMutableArray *section1 = [self.arrayOfSections objectAtIndex:0];
  NSMutableArray *section2 = [self.arrayOfSections objectAtIndex:1];

  NSString *cell2InSection1 = [section1 objectAtIndex:1];
  [section1 removeObject:cell2InSection1];

  [section2 insertObject:cell2InSection1
                 atIndex:0];

  NSIndexPath *sourceIndexPath = [NSIndexPath indexPathForRow:1
                                                    inSection:0];
  NSIndexPath *destinationIndexPath = [NSIndexPath indexPathForRow:0
                                                         inSection:1];

  [self.myTableView moveRowAtIndexPath:sourceIndexPath
                           toIndexPath:destinationIndexPath];

}
```

Результаты перехода показаны на рис. 4.14.

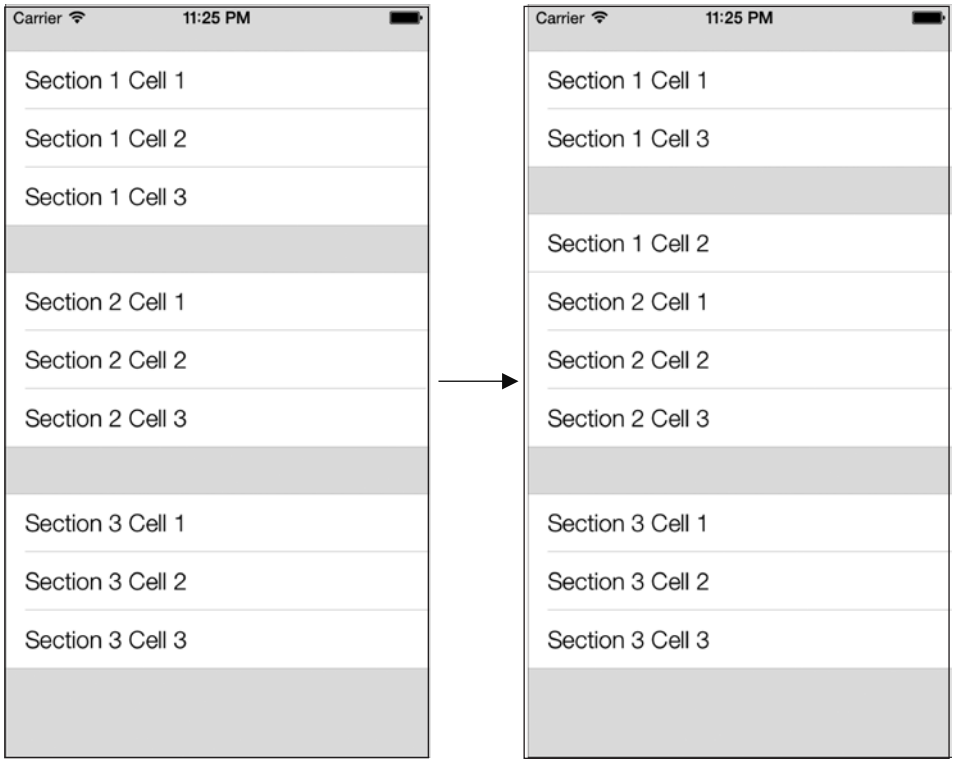


**Рис. 4.14.** Ячейка 2 из раздела 1 перемещена на место ячейки 1 из раздела 2

# 4.8. Удаление ячеек и разделов в табличных видах

## Постановка задачи

Требуется удалять из табличных видов разделы и/или ячейки, сопровождая этот процесс анимацией.

## Решение

Для удаления разделов из табличного вида выполните следующие шаги.

1. Сначала удалите раздел (-ы) в источнике данных независимо от того, с какой именно моделью данных вы работаете — Core Data или словарь/массив.
2. Примените к табличному виду метод экземпляра `deleteSections:withRowAnimation:`, относящийся к `UITableView`. Первый параметр, который нужно передать данному методу, имеет тип `NSIndexSet`. Этот объект можно инстанцировать с помощью метода класса `indexSetWithIndex:`, относящегося к классу `NSIndexSet`, где указываемый индекс — это беззнаковое целое число. Применяя такой подход, вы можете удалять только один раздел за раз. Если вы собираетесь удалить за раз более одного раздела, пользуйтесь методом класса `indexSetWithIndexesInRange:`, также относящимся к классу `NSIndexSet`, чтобы создать индексное множество с указанием диапазона. Это индексное множество передается описанному ранее методу экземпляра, относящемуся к `UITableView`.

Если вы хотите удалить ячейки в табличном виде, выполните следующие шаги.

1. Сначала удалите ячейку (ячейки) из источника данных. Здесь также не имеет значения, работаете ли вы с Core Data, обычным словарем, массивом или чем-то еще. Самое важное в данном случае — удалить из источника данных те объекты, которые соответствуют ячейкам табличного вида.
2. Теперь для удаления самих ячеек, соответствующих объектам данных, примените метод экземпляра `deleteRowsAtIndexPaths:withRowAnimation:`, относящийся к табличному виду. Первый параметр, который необходимо передать данному методу, — это массив типа `NSArray`. Данный массив должен содержать объекты типа `NSIndexPath`, и каждый индексный путь представляет одну ячейку в табличном виде. В каждом индексном пути содержится указание на раздел и на строку табличного вида. Этот путь составляется с помощью метода класса `indexPathForRow:inSection:`, относящегося к классу `NSIndexPath`.

## Обсуждение

В коде вашего пользовательского интерфейса вам может понадобиться удалять ячейки и/или разделы. Например, у вас может иметься переключатель (типа `UISwitch`, см. раздел 1.2). Когда пользователь нажимает переключатель, вам, возможно, требуется добавить в табличный вид несколько строк. После того как пользователь вернет переключатель в исходное положение, вам, вероятно, потребуется вновь убрать эти

строки с экрана. Но такие операции удаления не всегда ограничиваются ячейками (строками) табличного вида. Иногда из табличного вида требуется одновременно удалить целый раздел (или несколько разделов). Ключевой аспект при удалении разделов и ячеек из табличных видов состоит в том, что сначала из источника данных удаляется информация, соответствующая этим элементам (ячейкам или разделам), а потом вызываются соответствующие методы удаления, применяемые к табличному виду. После того как метод удаления завершит работу, табличный вид снова будет ссылаться на свой объект из источника данных. Если же после операции удаления количество ячеек/разделов в источнике данных не совпадет с количеством ячеек/разделов в табличном виде, приложение аварийно завершится. Но не волнуйтесь — даже если вы и допустите такую ошибку, то отладочное сообщение, которое появится на консоли, будет достаточно подробным для того, чтобы вы могли подправить код.

Рассмотрим, как удалять разделы из табличного вида. В данном разделе мы отобразим табличный вид в контроллере вида, который, в свою очередь, будет находиться в навигационном контроллере. Внутри табличного вида будет два раздела: один для нечетных чисел, другой — для четных. В табличном виде в разделе с нечетными числами мы отобразим только 1, 3, 5 и 7, а в разделе с четными — 0, 2, 4 и 6. В первом упражнении мы собираемся создать на навигационной панели специальную кнопку, которая будет удалять раздел с нечетными числами. На рис. 4.15 показано, какой результат мы хотим получить.

Начнем с главного. Определим контроллер вида:

```
#import <UIKit/UIKit.h>
static NSString *CellIdentifier = @"NumbersCellIdentifier";

@interface ViewController : UIViewController <UITableViewDelegate,
                                              UITableViewDataSource>

@property (nonatomic, strong) UITableView *tableViewNumbers;
@property (nonatomic, strong) NSMutableDictionary *dictionaryOfNumbers;
@property (nonatomic, strong) UIBarButtonItem *barButtonAction;

@end
```

Свойство `tableViewNumbers` соответствует нашему табличному виду. Свойство `barButtonAction` соответствует кнопке для удаления, которая будет отображаться на навигационной панели. И последнее, но немаловажное свойство `dictionaryOfNumbers` — это источник данных для табличного вида. В данном словаре мы поместим два значения типа `NSMutableArray`, которые будут содержать числа типа `NSNumber`. Это изменяемые массивы, позже в данной главе мы сможем удалять их отдельно от массивов, содержащихся в словаре. Ключи для этих массивов мы будем хранить как статические значения в файле реализации контроллера вида. По этой причине позже просто сможем извлечь массивы из словаря, пользуясь статическими ключами. (Если бы ключи не были статическими, то для нахождения массивов в словаре пришлось бы выполнять сравнение строк. А эта операция требует больше времени, чем обычное ассоциирование объекта со статическим ключом, не изменяющимся на протяжении всего существования контроллера вида.) Теперь синтезируем наши свойства и определим статические строковые ключи для массивов, находящихся в словаре источника данных:

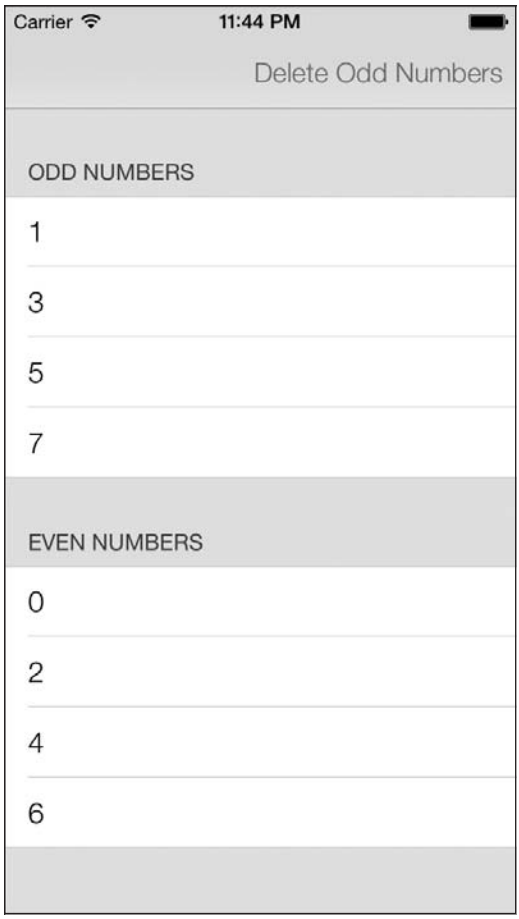


**Рис. 4.15.** Пользовательский интерфейс для отображения двух разделов табличного вида; в интерфейсе есть кнопка, удаляющая раздел Odd Numbers (Нечетные числа)

```
static NSString *SectionOddNumbers = @"Odd Numbers";
static NSString *SectionEvenNumbers = @"Even Numbers";

@implementation ViewController
```

Теперь, перед тем как создать табличный вид, необходимо заполнить информацией словарь источника данных. Вот простой метод, который автоматически заполнит словарь:

```
- (NSMutableDictionary *) dictionaryOfNumbers{

  if (_dictionaryOfNumbers == nil){
    NSMutableArray *arrayOfEvenNumbers =
    [[NSMutableArray alloc] initWithArray:@[
                                          @0,
                                          @2,
```

```
                                          @4,
                                          @6,
                                          ]];
    NSMutableArray *arrayOfOddNumbers =
    [[NSMutableArray alloc] initWithArray:@[
                                          @1,
                                          @3,
                                          @5,
                                          @7,
                                          ]];
    _dictionaryOfNumbers =
    [[NSMutableDictionary alloc]
     initWithDictionary:@{
                          SectionEvenNumbers : arrayOfEvenNumbers,
                          SectionOddNumbers : arrayOfOddNumbers,
                          }];
  }
  return _dictionaryOfNumbers;
}
```

Пока все нормально? Как видите, у нас два массива, в каждом из которых содержатся некоторые числа (в одном нечетные, в другом — четные). Мы ассоциируем массивы с ключами `SectionEvenNumbers` и `SectionOddNumbers`, которые ранее определили в файле реализации контроллера вида. Теперь инстанцируем табличный вид:

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

  self.barButtonAction =
  [[UIBarButtonItem alloc]
   initWithTitle:@"Delete Odd Numbers"
           style:UIBarButtonItemStylePlain
          target:self
          action:@selector(deleteOddNumbersSection:)];
  [self.navigationItem setRightBarButtonItem:self.barButtonAction
                                    animated:NO];

  self.tableViewNumbers = [[UITableView alloc]
                           initWithFrame:self.view.frame
                           style:UITableViewStyleGrouped];
  self.tableViewNumbers.autoresizingMask = UIViewAutoresizingFlexibleWidth |
                                           UIViewAutoresizingFlexibleHeight;
  self.tableViewNumbers.delegate = self;
  self.tableViewNumbers.dataSource = self;
  [self.view addSubview:self.tableViewNumbers];

}
```

Далее нужно заполнить табличный вид информацией внутри словаря источника с данными:

```
- (NSInteger) numberOfSectionsInTableView:(UITableView *)tableView{

  return self.dictionaryOfNumbers.allKeys.count;

}

- (NSInteger) tableView:(UITableView *)tableView
  numberOfRowsInSection:(NSInteger)section{

  NSString *sectionNameInDictionary =
    self.dictionaryOfNumbers.allKeys[section];

  NSArray *sectionArray = self.dictionaryOfNumbers[sectionNameInDictionary];
  return sectionArray.count;

}

- (UITableViewCell *) tableView:(UITableView *)tableView
         cellForRowAtIndexPath:(NSIndexPath *)indexPath{

  UITableViewCell *cell = nil;

  cell = [tableView dequeueReusableCellWithIdentifier:CellIdentifier
                                         forIndexPath:indexPath];

  NSString *sectionNameInDictionary =
    self.dictionaryOfNumbers.allKeys[indexPath.section];

  NSArray *sectionArray = self.dictionaryOfNumbers[sectionNameInDictionary];

  NSNumber *number = sectionArray[indexPath.row];

  cell.textLabel.text = [NSString stringWithFormat:@"%lu",
                         (unsigned long)[number unsignedIntegerValue]];

  return cell;

}

- (NSString *)  tableView:(UITableView *)tableView
  titleForHeaderInSection:(NSInteger)section{

  return self.dictionaryOfNumbers.allKeys[section];

}
```

Навигационная кнопка связана с селектором `deleteOddNumbersSection:`. Этот метод нам сейчас предстоит запрограммировать. Цель метода, как видно из его названия[1], — найти раздел, соответствующий всем нечетным числам в источнике

[1] Odd (англ.) — «нечетный», delete (англ.) — «удалить». — *Примеч. пер.*

данных, найти табличный вид, а потом удалить искомый раздел и из таблицы, и из источника данных. Вот как это делается:

```
- (void) deleteOddNumbersSection:(id)paramSender{

  /* Сначала удаляем раздел из источника данных. */
  NSString *key = SectionOddNumbers;
  NSInteger indexForKey = [[self.dictionaryOfNumbers allKeys]
                           indexOfObject:key];
  if (indexForKey == NSNotFound){
    NSLog(@"Could not find the section in the data source.");
    return;
  }
  [self.dictionaryOfNumbers removeObjectForKey:key];

  /* Затем удаляем раздел из табличного вида. */
  NSIndexSet *sectionToDelete = [NSIndexSet indexSetWithIndex:indexForKey];
  [self.tableViewNumbers deleteSections:sectionToDelete
                       withRowAnimation:UITableViewRowAnimationAutomatic];

  /* Наконец, убираем с навигационной панели кнопку,
   так как она нам больше не понадобится. */
  [self.navigationItem setRightBarButtonItem:nil animated:YES];

}
```

Все довольно просто. Теперь, когда пользователь нажмет кнопку на навигационной панели, раздел Odd Numbers (Нечетные числа) исчезнет из табличного вида. Как видите, в процессе удаления раздела табличный вид анимируется. Это происходит потому, что мы передали анимационный тип `UITableViewRowAnimationAutomatic` параметру `withRowAnimation:` метода `deleteSections:withRowAnimation:` табличного вида. Теперь запустите приложение в эмуляторе iOS и выполните Debug ▸ Toggle Slow Animations (Отладка ▸ Включить медленную анимацию). Потом попробуйте нажать кнопку на навигационной панели и посмотрите, что происходит. Как видите, удаление сопровождается медленной анимацией (движением). Красиво, правда? Когда удаление завершится, приложение будет выглядеть, как на рис. 4.16.

Вы уже знаете, как удалять разделы из табличных видов. Перейдем к удалению ячеек. Мы собираемся изменить функциональность навигационной кнопки так, чтобы при ее нажатии во всех разделах табличного вида удалялись все ячейки, содержащие числовое значение больше 2. Таким образом, мы удалим все четные и нечетные числа больше 2. Итак, изменим навигационную кнопку в методе `viewDidLoad` контроллера вида:

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

  self.barButtonAction =
  [[UIBarButtonItem alloc]
   initWithTitle:@"Delete Numbers > 2"
           style:UIBarButtonItemStylePlain
```


**Рис. 4.16.** Раздел, содержащий нечетные числа, удален из табличного вида

```
          target:self
          action:@selector(deleteNumbersGreaterThan2:)];
  [self.navigationItem setRightBarButtonItem:self.barButtonAction
                                    animated:NO];

  self.tableViewNumbers = [[UITableView alloc]
                           initWithFrame:self.view.frame
                           style:UITableViewStyleGrouped];
  self.tableViewNumbers.autoresizingMask =
    UIViewAutoresizingFlexibleWidth |
    UIViewAutoresizingFlexibleHeight;
  self.tableViewNumbers.delegate = self;
  self.tableViewNumbers.dataSource = self;
  [self.view addSubview:self.tableViewNumbers];

}
```

На рис. 4.17 показано, как выглядит приложение при запуске в эмуляторе iPhone.

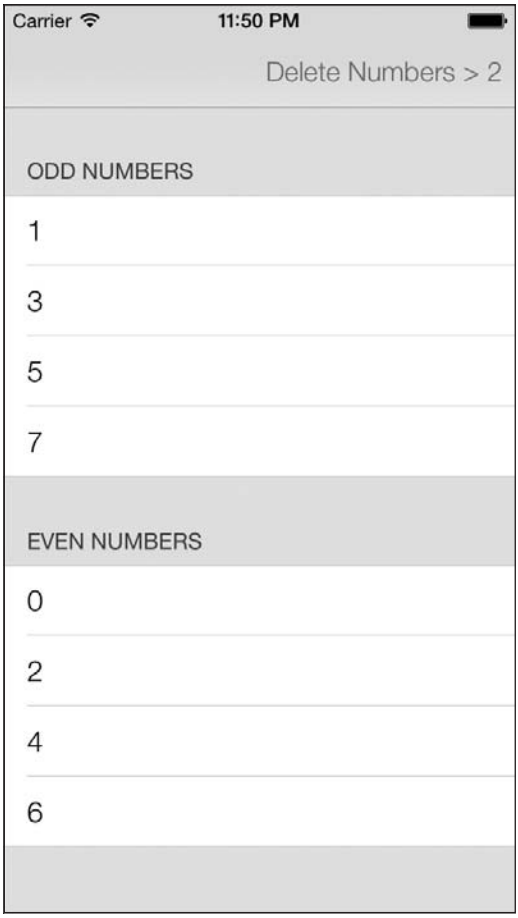


**Рис. 4.17.** Кнопка, удаляющая все ячейки с числами больше 2

Теперь кнопка навигационной панели связана с селектором `deleteNumbersGreaterThan2:`. Селектор — это метод, реализованный в контроллере вида. Но прежде, чем перейти к его программированию, определим, что этот метод должен сделать.

1. Найти оба массива с нечетными и четными числами в источнике данных и собрать индексные пути (типа `NSIndexPath`) чисел больше 2. Позже мы будем пользоваться этими индексными путями для удаления соответствующих ячеек в табличном виде.
2. Удалить все числа больше 2 из источника данных — как из словаря для нечетных чисел, так и из словаря для четных.
3. Удалить из табличного вида соответствующие ячейки. Индексные пути к этим ячейкам мы собрали на первом этапе.

4. Удалить кнопку с навигационной панели. Эта кнопка больше не понадобится, ведь ячейки уже удалены и из источника данных, и из табличного вида. В качестве альтернативы при желании можете просто отключить эту кнопку. Но мне кажется, что для удобства пользователя кнопку лучше просто удалить, поскольку отключенная кнопка все равно будет ему совершенно бесполезна.

```
- (void) deleteNumbersGreaterThan2:(id)paramSender{

  NSMutableArray *arrayOfIndexPathsToDelete =
    [[NSMutableArray alloc] init];
  NSMutableArray *arrayOfNumberObjectsToDelete =
    [[NSMutableArray alloc] init];

  /* Шаг 1: собираем объекты, которые мы хотим удалить из
     источника данных, а также их индексные пути. */
  __block NSUInteger keyIndex = 0;
  [self.dictionaryOfNumbers enumerateKeysAndObjectsUsingBlock:
   ^(NSString *key, NSMutableArray *object, BOOL *stop) {

     [object enumerateObjectsUsingBlock:
      ^(NSNumber *number, NSUInteger numberIndex, BOOL *stop) {

        if ([number unsignedIntegerValue] > 2){
          NSIndexPath *indexPath =
            [NSIndexPath indexPathForRow:numberIndex
                               inSection:keyIndex];
          [arrayOfIndexPathsToDelete addObject:indexPath];
          [arrayOfNumberObjectsToDelete addObject:number];
        }

     }];

     keyIndex++;
  }];

  /* Шаг 2: удаляем объекты из источника данных. */
  if ([arrayOfNumberObjectsToDelete count] > 0){
      NSMutableArray *arrayOfOddNumbers =
        self.dictionaryOfNumbers[SectionOddNumbers];
      NSMutableArray *arrayOfEvenNumbers =
        self.dictionaryOfNumbers[SectionEvenNumbers];
      [arrayOfNumberObjectsToDelete enumerateObjectsUsingBlock:
       ^(NSNumber *numberToDelete, NSUInteger idx, BOOL *stop) {
         if ([arrayOfOddNumbers indexOfObject:numberToDelete]
              != NSNotFound){
             [arrayOfOddNumbers removeObject:numberToDelete];
         }
         if ([arrayOfEvenNumbers indexOfObject:numberToDelete]
              != NSNotFound){
             [arrayOfEvenNumbers removeObject:numberToDelete];
         }
```

```
        }];
    }
    /* Шаг 3: удаляем все ячейки, соответствующие объектам. */
    [self.tableViewNumbers
        deleteRowsAtIndexPaths:arrayOfIndexPathsToDelete
        withRowAnimation:UITableViewRowAnimationAutomatic];
    [self.navigationItem setRightBarButtonItem:nil animated:YES];
}
```

После того как пользователь нажмет кнопку на навигационной панели, все ячейки, в которых содержатся числа больше 2, будут удалены из источника данных. Табличный вид и все приложение станут выглядеть как на рис. 4.18.

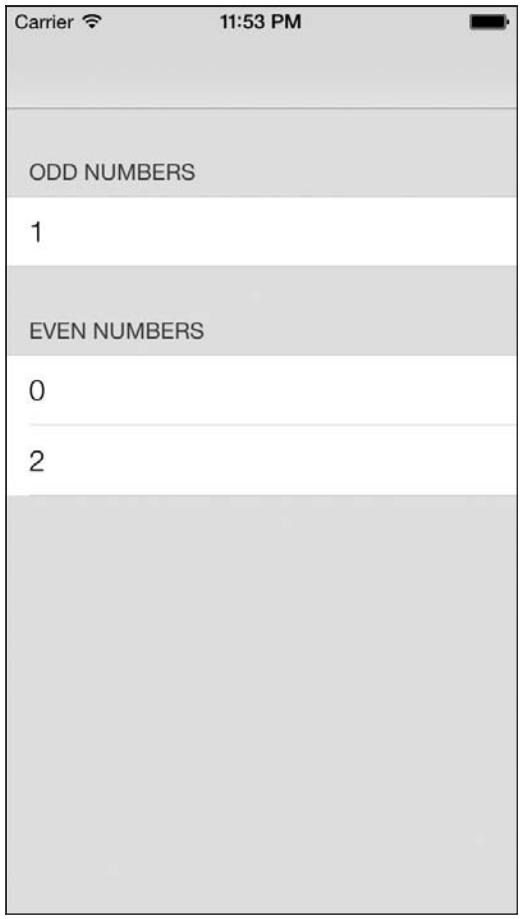


**Рис. 4.18.** Мы удалили все ячейки, в которых содержались числа больше 2

## См. также

Раздел 1.2.

# 4.9. Использование UITableViewController для удобства при создании табличных видов

## Постановка задачи

Требуется возможность быстро создавать табличные виды.

## Решение

Используйте контроллер вида `UITableViewController`, который по умолчанию предоставляется с табличным контроллером вида.

## Обсуждение

В инструментарии iOS SDK есть очень удобный класс `UITableViewController`, который предоставляется с заранее заготовленным экземпляром табличного вида. Чтобы пользоваться всеми его преимуществами, всего лишь потребуется создать новый класс, наследующий от указанного. Здесь я подробно опишу все этапы создания нового проекта Xcode, использующего табличный контроллер вида.

1. На панели меню Xcode выберите File ▸ New ▸ Project (Файл ▸ Новый ▸ Проект).
2. Убедитесь, что в левой части экрана выбрана категория iOS. Затем перейдите в подкатегорию Application (Приложение). В правой части экрана выберите шаблон Empty Application (Пустое приложение), а потом нажмите кнопку Next (Далее) (рис. 4.19).
3. На следующем экране просто выберите название для вашего проекта. Кроме того, убедитесь, что вся информация у вас на экране, кроме Organization Name (Название организации) и Company Identifier (Идентификатор компании), в точности соответствует той, что приведена на рис. 4.20. Как только все будет готово, нажмите кнопку Next (Далее).
4. На следующем экране вам будет предложено сохранить приложение на диске. Просто сохраните приложение в месте, которое кажется вам целесообразным, и нажмите кнопку Create (Создать).
5. В Xcode выберите меню File ▸ New ▸ File (Файл ▸ Новый ▸ Файл).
6. Убедитесь, что в левой части диалогового окна категория iOS выбрана в качестве основной и при этом также выбрана подкатегория Cocoa Touch. Далее в правой части диалогового окна выберите класс Objective-C (рис. 4.21).
7. На следующем экране вам будет предложено выбрать суперкласс для нового класса. Это очень важный этап. Убедитесь, что в качестве суперкласса задан `UITableViewController`. Удостоверьтесь, что все остальные настройки у вас точно такие же, как и у меня на рис. 4.22. Когда все будет готово, нажмите кнопку Next (Далее).

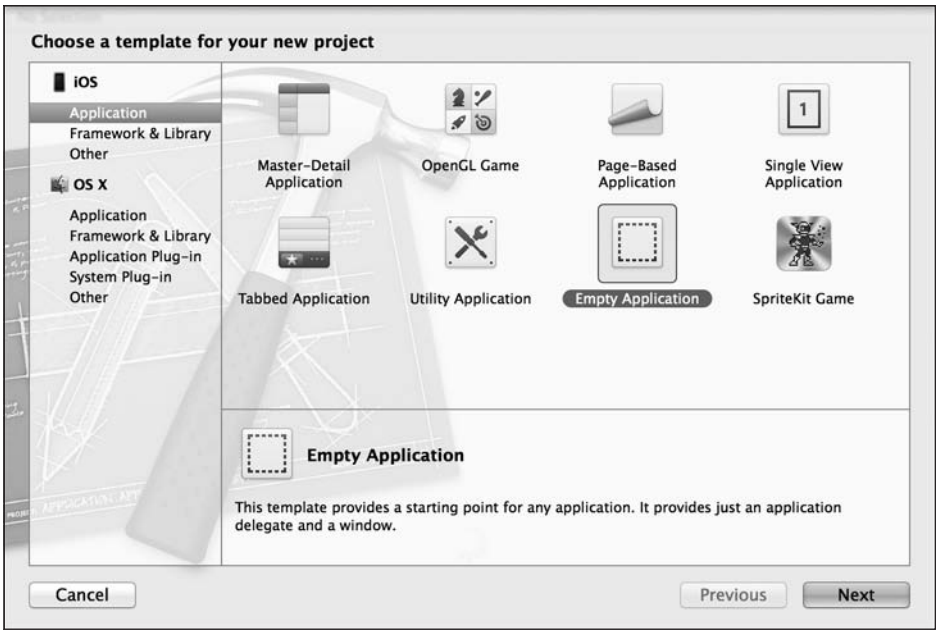


**Рис. 4.19.** Создание нового пустого приложения, в котором позже будет находиться табличный контроллер

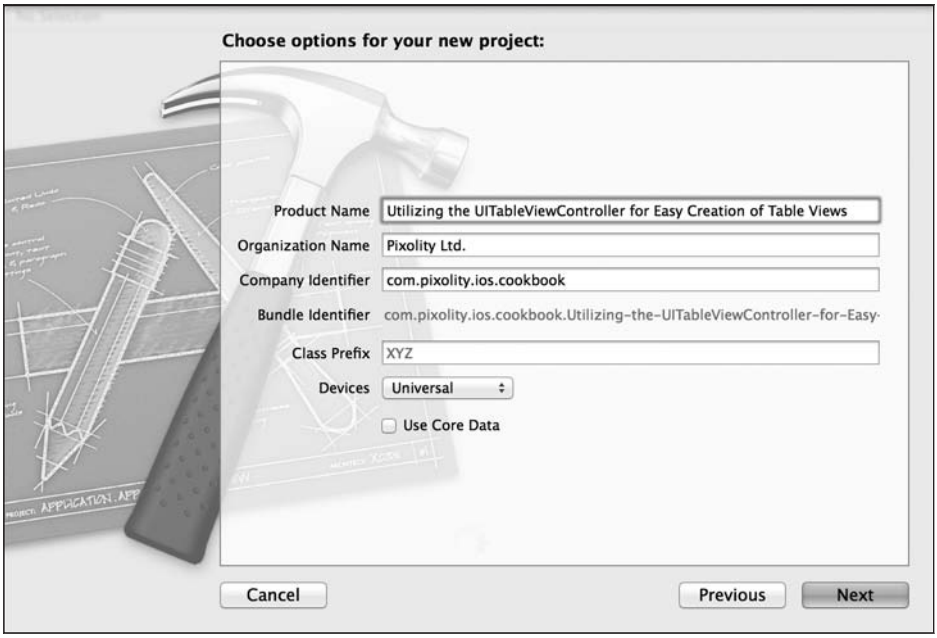


**Рис. 4.20.** Конфигурирование нового пустого приложения в Xcode

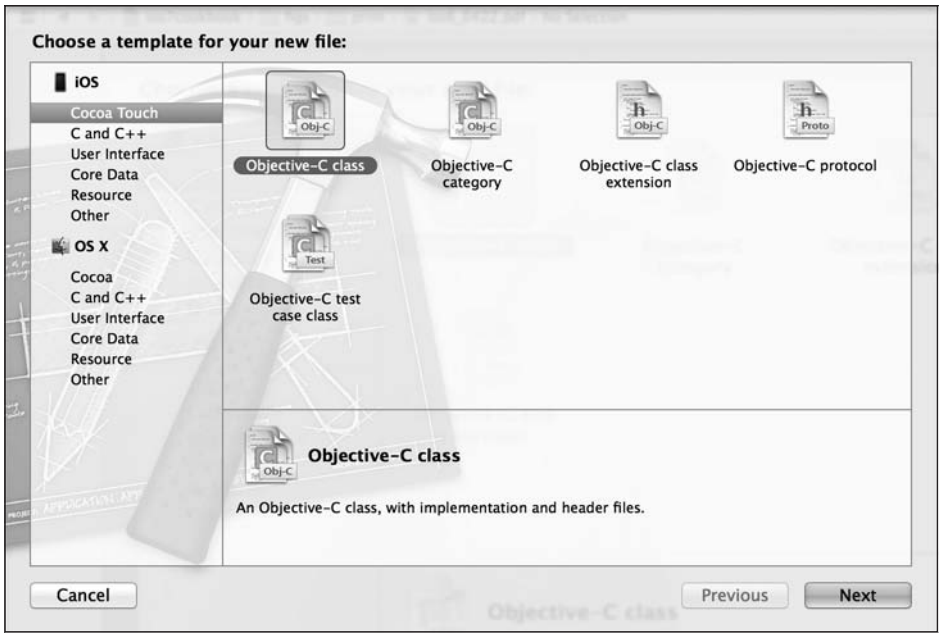


**Рис. 4.21.** Создание нового класса для табличного вида с контроллером

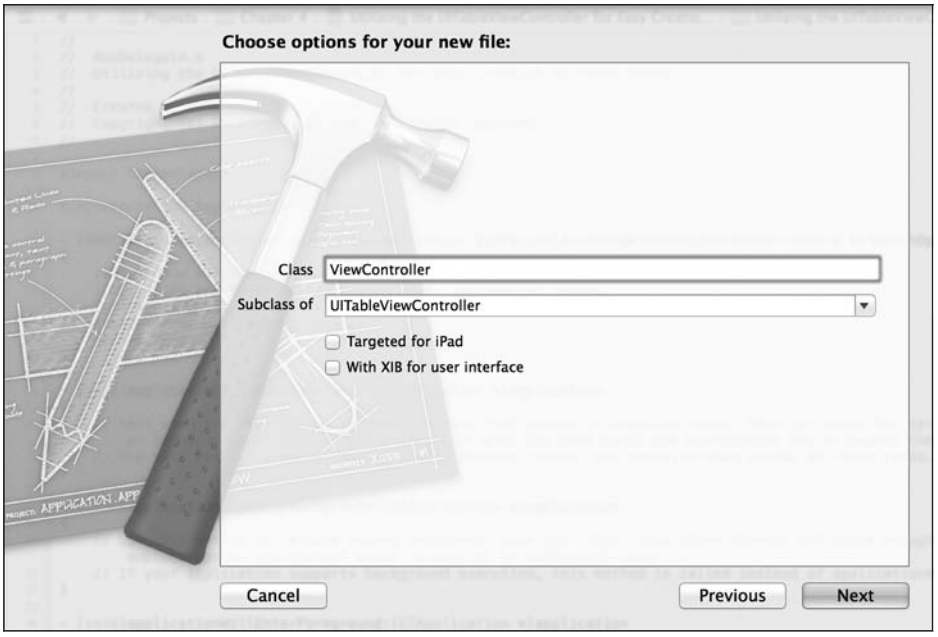


**Рис. 4.22.** Задаем суперкласс для нового объекта, который станет контроллером табличного вида

8. На следующем экране вы сможете сохранить табличный контроллер вида в проекте. Сохраните его как класс `ViewController` и нажмите кнопку **Create** (Создать).
9. В файле реализации делегата вашего приложения обязательно импортируйте заголовочный файл этого контроллера вида, а затем создайте экземпляр этого класса и установите его в качестве корневого контроллера вида приложения, как показано далее:

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

@implementation AppDelegate

- (BOOL) application:(UIApplication *)application
  didFinishLaunchingWithOptions:(NSDictionary *)launchOptions{

    ViewController *controller = [[ViewController alloc]
                                  initWithStyle:UITableViewStylePlain];

  self.window = [[UIWindow alloc]
                 initWithFrame:[[UIScreen mainScreen] bounds]];

  self.window.rootViewController = controller;

  self.window.backgroundColor = [UIColor whiteColor];
  [self.window makeKeyAndVisible];
  return YES;
}
```

Теперь, если вы попытаетесь скомпилировать проект, компилятор выдаст вам следующие предупреждения[1]:

```
ViewController.m:47:2: Potentially incomplete method implementation.
ViewController.m:54:2: Incomplete method implementation.
```

Итак, необходимо иметь в виду, что компилятор выдает определенные предупреждения, об устранении которых придется позаботиться в файле реализации контроллера вида. Открыв этот файл, вы увидите, что Apple вставила в шаблон класса табличного контроллера вида макрокоманды `#warning` — инструкции для компилятора (именно они приводят к тому, что на экран выводятся показанные ранее предупреждения). Одно из предупреждений находится в методе `numberOfSectionsInTableView:`, другое — в методе `tableView:numberOfRowsInSection:`. Мы видим на экране предупреждения, потому что не запрограммировали логику для этих методов. Минимальная информация, необходимая табличному контроллеру вида, — это количество разделов для отображения, количество строк для отображения, а также объект ячейки, который должен отображаться в каждой из строк. Мы не видим никаких предупреждений,

[1] Первая строка: «Потенциально неполная реализация метода». Вторая строка: «Неполная реализация метода». — *Примеч. пер.*

связанных с отсутствием реализации объекта ячейки, но только потому, что Apple по умолчанию предоставляет формальную реализацию этого метода, создающую за вас пустые ячейки.

По умолчанию контроллер табличного вида является источником данных и делегатом табличного вида. Вам не придется отдельно указывать источник данных и делегат для этого табличного вида.

Перейдем к файлу реализации табличного контроллера вида и убедимся, что у нас есть массив строк (просто для примера), которыми мы можем заполнить табличный вид:

```
#import "ViewController.h"

static NSString *CellIdentifier = @"Cell";

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

@implementation ViewController

- (id)initWithStyle:(UITableViewStyle)style
{
  self = [super initWithStyle:style];
  if (self) {
    // Специальная инициализация
    self.allItems = @[
                    @"Anthony Robbins",
                    @"Steven Paul Jobs",
                    @"Paul Gilbert",
                    @"Yngwie Malmsteen"
                    ];

    [self.tableView registerClass:[UITableViewCell class]
           forCellReuseIdentifier:CellIdentifier];

  }

  return self;
}

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

- (NSInteger)numberOfSectionsInTableView:(UITableView *)tableView{
  return 1;
}
```

```
- (NSInteger)tableView:(UITableView *)tableView
  numberOfRowsInSection:(NSInteger)section{
  return self.allItems.count;
}

- (UITableViewCell *)tableView:(UITableView *)tableView
         cellForRowAtIndexPath:(NSIndexPath *)indexPath{

    UITableViewCell *cell = [tableView
                             dequeueReusableCellWithIdentifier:CellIdentifier
                             forIndexPath:indexPath];

  cell.textLabel.text = self.allItems[indexPath.row];

  return cell;
}

@end
```

Теперь, запустив приложение, мы увидим результат, напоминающий рис. 4.23.

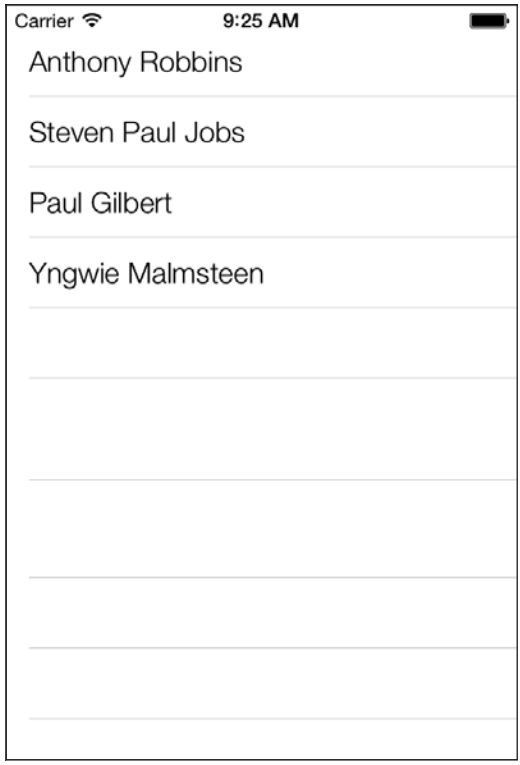


**Рис. 4.23.** Строки правильно отображаются в табличном виде

Вот практически и все, что следует знать о табличных контроллерах видов. Еще раз напомню, что табличный контроллер вида одновременно является *и* источником данных, *и* делегатом табличного вида. Итак, теперь вы можете реализовать методы протокола `UITableViewDataSource`, а также методы протокола `UITableViewDelegate` прямо в файле реализации табличного контроллера вида.

## См. также

Раздел 4.1.

# 4.10. Отображение элемента управления, предназначенного для обновления информации в табличных видах

## Постановка задачи

Требуется отображать в пользовательском интерфейсе красивый элемент управления для обновления информации. Этот элемент управления должен находиться над табличными видами и служить для пользователя интуитивно понятным инструментом: такой инструмент позволяет временно убрать таблицу с экрана, а потом вновь вывести ее, но уже с обновленной информацией. Пример показан на рис. 4.24.

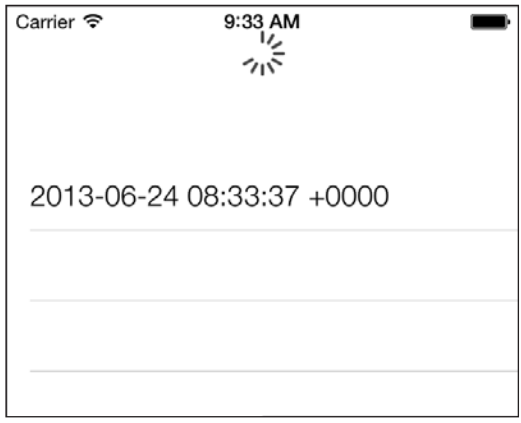


**Рис. 4.24.** Элемент управления для обновления информации, расположенный над табличным видом

## Решение

Создайте табличный контроллер вида (так, как описано в разделе 4.9) и задайте в качестве значения его свойства `refreshControl` новый экземпляр класса `UIRefreshControl`, как показано далее:

```
- (id)initWithStyle:(UITableViewStyle)style{
  self = [super initWithStyle:style];
  if (self) {

    [self.tableView registerClass:[UITableViewCell class]
           forCellReuseIdentifier:CellIdentifier];
    self.allTimes = [NSMutableArray arrayWithObject:[NSDate date]];

    /* Создаем элемент управления для обновления информации */
    self.refreshControl = [[UIRefreshControl alloc] init];
    self.refreshControl = self.refreshControl;
    [self.refreshControl addTarget:self
                            action:@selector(handleRefresh:)
                  forControlEvents:UIControlEventValueChanged];

  }
  return self;
}
```

## Обсуждение

Элементы управления для обновления информации — это простые визуальные индикаторы, располагающиеся над табличным видом и сообщающие пользователю, что какая-то информация в таблице сейчас обновится. Например, чтобы обновить содержимое почтового ящика в приложении Mail в версиях старше iOS 6, вам приходилось нажимать на специальную кнопку Refresh (Обновить). В новой iOS 7 вы можете просто потянуть список ваших писем вниз, как если бы хотели ознакомиться с какими-то письмами из верхней части списка, которые пока не успели прочитать. Как только iOS зафиксирует такой жест, система инициирует обновление. Круто, правда? Это нововведение впервые появилось в Twitter-клиенте для iPhone, большое спасибо за это его разработчикам. Apple по достоинству оценила всю элегантность и логичность этой возможности обновления видов, поэтому в SDK был добавлен специальный компонент для реализации такой функции. Класс, соответствующий этому компоненту, называется `UIRefreshControl`.

Чтобы создать новый экземпляр этого класса, достаточно просто вызвать его метод `init`. Сделав это, добавьте экземпляр к табличному контроллеру вида, как описано в подразделе «Решение» данного раздела.

Итак, вы хотите знать, когда пользователь инициирует обновление информации в табличном виде. Для этого просто вызовите метод экземпляра `addTarget:action:forControlEvents:` обновляющего элемента и передайте ему целевой объект вместе с селектором этого объекта — остальное система сделает за вас. Передайте событие `UIControlEventValueChanged` параметру `forControlEvents` этого метода.

Сейчас я это продемонстрирую. В данном примере имеем табличный контроллер вида, в котором отображаются дата и время в строковом формате. Как только пользователь обновит список, потянув его вниз, мы будем добавлять сюда новые актуальные значения даты и времени, изменяя таким образом таблицу. Итак, всякий раз, когда пользователь опускает таблицу, инициируется

обновление, в ходе которого мы можем добавить в список актуальные значения даты и времени, обновив табличный вид. Итак, начнем с файла реализации контроллера вида. Определим элемент управления для обновления информации и источник данных:

```
#import "ViewController.h"

static NSString *CellIdentifier = @"Cell";
@interface ViewController ()
@property (nonatomic, strong) NSMutableArray *allTimes;
@property (nonatomic, strong) UIRefreshControl *refreshControl;
@end

@implementation ViewController
```

Свойство `allTimes` — это обычный изменяемый массив, который будет содержать все экземпляры `NSDate` в момент, когда завершится обновление таблицы. Мы уже рассмотрели инициализацию табличного контроллера вида в подразделе «Решение» данного раздела, поэтому я не буду вновь писать об этом. Но, как вы помните, мы прикрепили событие `UIControlEventValueChanged` обновляющего элемента управления к методу `handleRefresh:`. В этом методе мы всего лишь собираемся добавить к массиву дату и время, после чего обновить табличный вид:

```
- (void) handleRefresh:(id)paramSender{

  /* Оставляем небольшую задержку между высвобождением обновляющего элемента
   управления и самим моментом обновления. Так весь процесс выглядит
   в интерфейсе более плавно, чем при использовании обычной анимации */
  int64_t delayInSeconds = 1.0f;
  dispatch_time_t popTime =
    dispatch_time(DISPATCH_TIME_NOW, delayInSeconds * NSEC_PER_SEC);

  dispatch_after(popTime, dispatch_get_main_queue(), ^(void){

    /* Добавляем актуальную дату к имеющемуся списку дат;
     Таким образом, при обновлении табличного вида новый информационный
     элемент на экране будет находиться над старым и пользователь увидит
     разницу во времени до и после обновления */
  [self.allTimes addObject:[NSDate date]];

  [self.refreshControl endRefreshing];

  NSIndexPath *indexPathOfNewRow =
    [NSIndexPath indexPathForRow:self.allTimes.count-1 inSection:0];
  [self.tableView
   insertRowsAtIndexPaths:@[indexPathOfNewRow]
         withRowAnimation:UITableViewRowAnimationAutomatic];
  });

}
```

Последний важный момент: мы записываем дату в табличный вид посредством методов делегата и источника данных табличного вида:

```
- (NSInteger)numberOfSectionsInTableView:(UITableView *)tableView{
  return 1;
}

- (NSInteger) tableView:(UITableView *)tableView
  numberOfRowsInSection:(NSInteger)section{
  return self.allTimes.count;
}

- (UITableViewCell *)tableView:(UITableView *)tableView
  cellForRowAtIndexPath:(NSIndexPath *)indexPath{

  UITableViewCell *cell = [tableView
                           dequeueReusableCellWithIdentifier:CellIdentifier
                           forIndexPath:indexPath];

  cell.textLabel.text = [NSString stringWithFormat:@"%@",
                         self.allTimes[indexPath.row]];

  return cell;
}
```

Опробуйте в эмуляторе или на устройстве то, что у нас получилось. Открыв приложение, вы сразу заметите только одно значение даты и времени в списке. Но если потянуть таблицу вниз, то постепенно перед вами будут открываться новые элементы (см. рис. 4.24).

## См. также

Раздел 4.9.

# 5 Выстраивание сложных макетов с помощью сборных видов

## 5.0. Введение

Табличные виды очень хороши. Действительно. Тем не менее они отличаются удручающей негибкостью, так как их содержимое всегда ориентировано по вертикали. Это не настоящие таблицы-сетки, поэтому они и функционально не похожи на сетки. Однако программист может оказаться в ситуации, когда требуется отрисовать на экране таблицеподобный компонент со строками и столбцами, а затем поместить в каждую из ячеек различные элементы пользовательского интерфейса и каждый элемент сделать интерактивным. В табличном виде у вас фактически имеется всего один столбец с множеством строк. Если вы хотите создать иллюзию множества столбцов, то придется предоставить собственную специальную ячейку и оформить ее так, как будто она состоит из нескольких столбцов.

Сборные виды, так же как табличные виды, состоят из ячеек, причем каждая ячейка содержит элемент или вид, отображаемый на экране. Ячейки в сборных видах доступны для повторного использования, причем их можно изымать из очереди и возвращать на экран в любой момент, когда это можно и нужно. Но компоновка страницы может быть практически любой вообразимой на двухмерном экране.

Именно поэтому в 6-й версии iOS компания Apple впервые внедрила сборные виды. Сборный вид можно сравнить с сильно усовершенствованным прокручиваемым видом. У него есть источник данных и делегат, как и у табличного вида. Но он обладает одним свойством, делающим его совершенно несхожим с табличным или прокручиваемым видами. Речь идет о *макетном объекте*.

В сущности, макетный объект вычисляет, где должен быть размещен каждый элемент, входящий в состав сборного вида. Однако Apple *немного* усложнила такую работу, внедрив конкретный класс для работы со сборными видами, причем этот класс нельзя инстанцировать напрямую. Вместо этого придется инстанцировать подкласс от этого класса, называемый `UICollectionViewFlowLayout`.

Этот подкласс обеспечивает последовательную компоновку, при которой ячейки из сборного вида распределяются на экране по секциям. Каждая секция — это группа ячеек сборного вида, так же как в табличном виде. Однако в сборном виде любая секция может компоноваться на экране разными способами, не обязательно вертикально. Например, у вас может быть три прямоугольника, в каждом из которых содержится своя маленькая таблица (рис. 5.1).

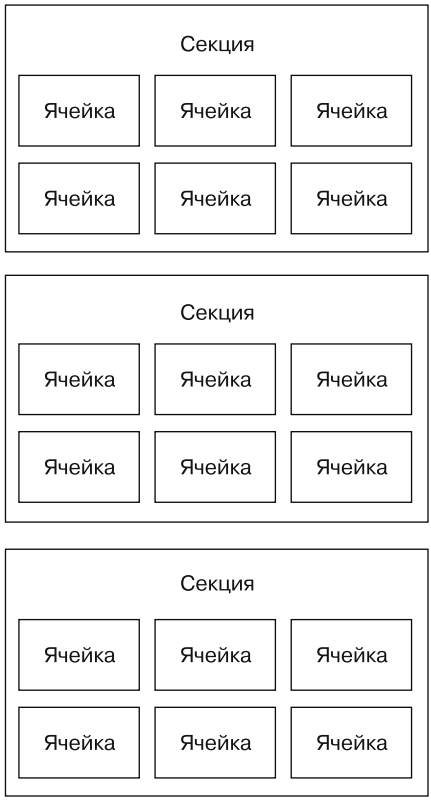


**Рис. 5.1.** Типичный макет с последовательной компоновкой в сборном виде

Как правило, секции располагаются на экране в виде таблиц, то есть образуя строки и столбцы. Именно эта задача решается с помощью класса последовательной компоновки. Если вы хотите добиться еще большей свободы действий при компоновке, то попробуйте изменить свойства класса последовательной компоновки. А если желаете сделать нечто, значительно отличающееся от стандартных возможностей последовательной компоновки, создайте для этого собственный класс. Например, такой специальный класс вам потребуется для создания сборного вида, который показан на рис. 5.2. Далее приведен специальный класс компоновки, располагающий соответствующие ячейки совсем не по табличному принципу.

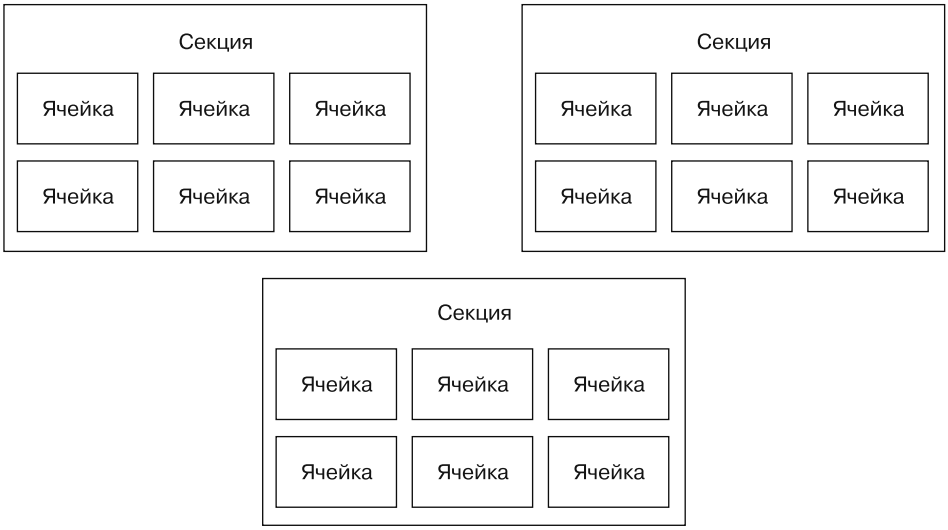


**Рис. 5.2.** Специальный вариант компоновки для сборного вида

# 5.1. Создание сборных видов

## Постановка задачи

Требуется отобразить на экране сборный вид.

## Решение

Либо воспользуйтесь экземпляром `UICollectionView`, который в таком случае нужно сделать дочерним видом одного из видов вашего приложения (если хотите создать полноэкранный сборный вид), либо примените класс `UICollectionViewController`.

## Обсуждение

Сборный вид, как и табличный, — это элемент, который может быть добавлен в качестве дочернего к другому виду. Итак, создавая приложение, определитесь с тем, должен ли сборный вид быть основным видом в контроллере или представлять собой небольшой фрагмент другого вида.

Сначала рассмотрим вариант с полноэкранным видом.

1. Откройте Xcode.
2. В меню File (Файл) выберите New (Новый), а затем Project (Проект).
3. Слева в качестве основной категории выберите iOS, а под ней — Application (Приложение). В правой части экрана выберите Empty Application (Пустое приложение), после чего нажмите кнопку Next (Далее).

4. На следующем экране введите информацию о вашем проекте и убедитесь, что установлен флажок Use Automatic Reference Counting (Использовать автоматический подсчет ссылок) (рис. 5.3). Как только введете все необходимые значения, нажмите кнопку Next (Далее).
5. После этого вам будет предложено сохранить проект на диске. Выберите подходящее для этого место и нажмите кнопку Create (Создать).
6. Теперь, когда проект подготовлен, создайте в нем новый класс и назовите его `ViewController`. Этот класс должен наследовать от `UICollectionViewController`. Вам не понадобится `.xib`-файл для этого контроллера вида, поэтому откажитесь от этой возможности (рис. 5.4).

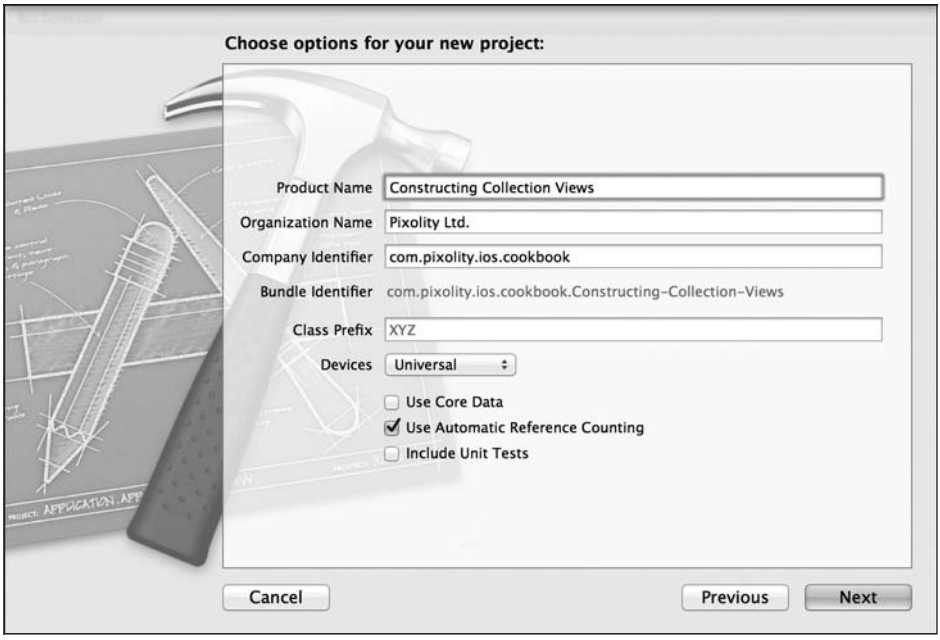


**Рис. 5.3.** Создание нового проекта Пустое приложение (Empty Application) для сборного вида

7. Найдите в проекте файл `AppDelegate.m` (это файл реализации делегата приложения) и откройте его, после чего создайте экземпляр сборного вида, а затем сделайте этот сборный вид корневым контроллером вида вашего приложения, как показано здесь:

```
- (BOOL) application:(UIApplication *)application
  didFinishLaunchingWithOptions:(NSDictionary *)launchOptions{

    /* Инстанцируем контроллер сборного вида с нулевым макетным объектом.
    Примечание: в результате программа выдаст исключение, но позже мы
    изучим, как создавать макетные объекты и предоставлять их нашим сборным
    видам */
    ViewController *viewController = [[ViewController alloc]
```

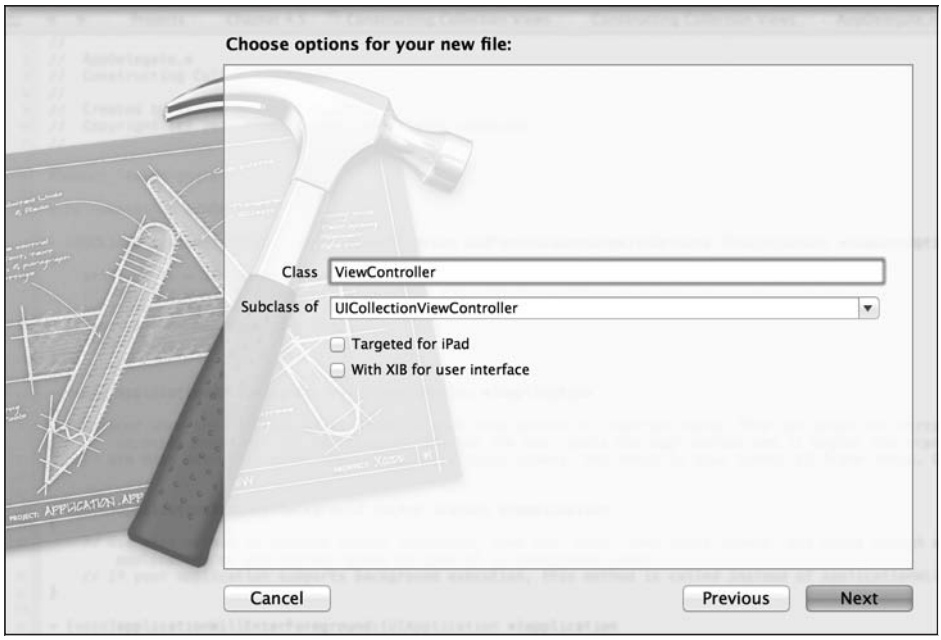


**Рис. 5.4.** Добавляем в проект новый класс сборного вида

```
                                        initWithCollectionViewLayout:nil];

    self.window = [[UIWindow alloc] initWithFrame:
                   [[UIScreen mainScreen] bounds]];

    self.window.backgroundColor = [UIColor whiteColor];

    /* Устанавливаем сборный вид в качестве корневого для нашего окна */
    self.window.rootViewController = viewController;
    [self.window makeKeyAndVisible];
    return YES;
}
```

Как только вы запустите ваше приложение, оно аварийно завершится и выдаст сообщение о том, что вы указали нулевой макетный объект для вашего сборного вида. И здесь среда времени исполнения действует совершенно правильно. Так делать нельзя. Но мы просто пока не обсудили, как инстанцировать макетные объекты и передавать их в сборные виды. Поэтому пока оставим все как есть. Далее в этой главе мы подробнее поговорим о макетных объектах сборных видов.

Итак, мы уже научились создавать контроллер сборного вида, и это хорошо, если вы хотите, чтобы такой вид при отображении занимал на устройстве весь экран. Однако если вы разрабатываете специальный компонент, входящий в состав другого, более крупного вида, то попробуйте просто инстанцировать объект типа

UICollectionView, воспользовавшись его выделенным инициализатором — методом initWithFrame:collectionViewLayout:.

Чтобы это сделать, требуется просто инстанцировать сборный вид, воспользовавшись указанным инициализатором. После инициализации вы сможете добавить сборный вид в качестве дочернего к другому виду. Например, если вы хотите добавить его к вашему виду контроллера вида, просто вызовите метод addSubview: этого вида с контроллером и передайте экземпляр вашего сборного вида этому методу в качестве параметра. Кроме того, нужно убедиться, что в качестве значений свойств delegate и dataSource сборного вида заданы валидные объекты, соответствующие протоколам UICollectionViewDelegate и UICollectionViewDataSource. Выполнить остальные операции не составляет труда. Далее в этой главе описаны все приемы, используемые для наполнения сборного вида информацией из источника данных и реагирования на события с помощью объекта-делегата.

## См. также

Раздел 5.0.

# 5.2. Присваивание источника данных сборному виду

## Постановка задачи

Требуется предоставить для сборного вида данные, которые будут выводиться на экран.

## Решение

Присвойте сборному виду источник данных, воспользовавшись свойством dataSource класса UICollectionView. Источник данных должен быть объектом, который соответствует протоколу UICollectionViewDataSource. Кроме того, само собой разумеется, что объект источника данных обязательно должен реализовывать методы и свойства этого протокола, относящиеся к категории required.

## Обсуждение

Источник данных сборного вида, как и источник данных табличного вида, отвечает за предоставление сборному виду достаточного для отображения на экране количества информации. Способ отображения данных на экране *не входит* в компетенцию источника данных — это задача макета. Ячейки, отображаемые макетным объектом в сборном виде, в итоге будут предоставляться источником данных сборного вида.

Вот методы протокола UICollectionViewDataSource, которые обязательно требуется реализовать в вашем источнике данных.

- collectionView:numberOfItemsInSection: — этот метод возвращает объект NSInteger, сообщающий сборному виду количество элементов, которые должны быть отображены в заданной секции. Задаваемая секция сообщается данному методу как целое число, представляющее собой индекс с нулевым основанием для данной секции. Именно так происходит и при работе с табличными видами.
- collectionView:cellForItemAtIndexPath: — ваша реализация этого метода должна возвращать экземпляр UICollectionViewCell, соответствующий ячейке, к которой ведет указанный индексный путь. Класс UICollectionViewCell наследует от UICollectionReusableView. Фактически любая доступная для повторного использования ячейка, передаваемая сборному виду для отображения, должна прямо или косвенно наследовать от UICollectionReusableView, о чем мы подробно поговорим в этой главе. Индексный путь указывается в параметре cellForItemAtIndexPath этого метода. Вы можете запросить индексы section и row этого элемента из индексного пути.

Перейдем к файлу реализации контроллера сборного вида (ViewController.m). Этот контроллер мы создали в разделе 5.1. Реализуем в данном файле рассмотренные ранее методы источника данных сборного вида:

```
#import "ViewController.h"

@implementation ViewController

/* Пока мы не собираемся возвращать никаких секций */
- (NSInteger)collectionView:(UICollectionView *)collectionView
  numberOfItemsInSection:(NSInteger)section{
    return 0;
}

/* Мы пока не знаем, как возвращать секции в сборный вид, поэтому для начала
возвратим здесь nil */
- (UICollectionViewCell *)collectionView:(UICollectionView *)collectionView
  cellForItemAtIndexPath:(NSIndexPath *)indexPath{
    return nil;
}

@end
```

*На данном этапе* этот код можно считать полным. Но, как было указано в разделе 5.1, при попытке его запустить приложение аварийно завершится. Дело в том, что делегат приложения устанавливает макетный объект сборного вида в значение nil. Эта проблема никуда не исчезла, мы собираемся устранить ее в разделе 5.3.

## См. также

Разделы 5.0 и 5.1.

# 5.3. Обеспечение последовательной компоновки в сборном виде

## Постановка задачи

Требуется, чтобы ваш сборный вид был скомпонован как таблица (сетка), чтобы его содержимое отображалось примерно так же, как на рис. 5.1.

## Решение

Создайте экземпляр класса `UICollectionViewFlowLayout`, инстанцируйте контроллер сборного вида с помощью выделенного метода-инициализатора `initWithCollectionViewLayout:` из класса `UICollectionViewController`, а затем передайте этому методу ваш макетный объект.

## Обсуждение

Макет для последовательной компоновки очень легко инстанцировать. Но прежде, чем можно будет передать его сборному виду, он должен быть сконфигурирован. Здесь мы обсудим различные свойства экземпляра класса `UICollectionViewFlowLayout` и поговорим о том, как их можно корректировать. Кроме того, нас интересует, как эти свойства влияют на отображение ячеек сборного вида на экране.

- `minimumLineSpacing` — значение с плавающей точкой, сообщающее макету с последовательной компоновкой минимальное количество точек, которые необходимо зарезервировать между рядами. Макетный объект может выделить и больше пространства, чтобы компоновка выглядела красиво, но меньше выделить не может. Если ваш сборный вид слишком мал и в него не помещаются все элементы, они будут обрезаться, как и любые другие виды в iOS SDK.
- `minimumInteritemSpacing` — значение с плавающей точкой, сообщающее макету с последовательной компоновкой минимальное количество точек, которые необходимо зарезервировать между ячейками в одной строке. Опять же это минимальное количество точек, и макет может увеличить это количество в зависимости от размера сборного вида.
- `itemSize` — величина `CGSize`, соответствующая размеру каждой ячейки в сборном виде.
- `scrollDirection` — значение типа `UICollectionViewScrollDirection`, сообщающее макету с последовательной компоновкой, как должно прокручиваться содержимое сборного вида. Содержимое может прокручиваться либо по горизонтали, либо по вертикали, но не в обоих направлениях одновременно. По умолчанию это свойство имеет значение `UICollectionViewScrollDirectionVertical`, но вы можете изменить его на `UICollectionViewScrollDirectionHorizontal`.
- `sectionInset` — значение типа `UIEdgeInsets`, задающее размер полей вокруг каждой секции. В принципе, поля — это пространство, которое не относится ни к одной

из ячеек. Для создания таких отступов можно воспользоваться функцией `UIEdgeInsetsMake`. У каждого поля есть верхний, нижний, правый и левый край, все они обозначаются числами с плавающей точкой. Не волнуйтесь, если это объяснение кажется путаным — вскоре все встанет на свои места.

В дальнейшем в этом разделе буду исходить из того, что вы уже выполнили инструкции, изложенные в разделах 5.1 и 5.2, и на данном этапе у вас есть приложение, в котором написан контроллер сборного вида, а также делегат приложения, отображающий этот контроллер сборного вида в качестве корневого контроллера вида окна. Теперь мы собираемся изменить делегат приложения, чтобы предоставить контроллеру сборного вида действующий механизм последовательной компоновки:

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

@implementation AppDelegate
- (UICollectionViewFlowLayout *) flowLayout{

  UICollectionViewFlowLayout *flowLayout =
  [[UICollectionViewFlowLayout alloc] init];

  flowLayout.minimumLineSpacing = 20.0f;
  flowLayout.minimumInteritemSpacing = 10.0f;
  flowLayout.itemSize = CGSizeMake(80.0f, 120.0f);
  flowLayout.scrollDirection = UICollectionViewScrollDirectionVertical;
  flowLayout.sectionInset = UIEdgeInsetsMake(10.0f, 20.0f, 10.0f, 20.0f);

  return flowLayout;
}

- (BOOL)            application:(UIApplication *)application
  didFinishLaunchingWithOptions:(NSDictionary *)launchOptions{

    /* Инстанцируем контроллер сборного вида с валидным макетом
    для последовательной компоновки */
    ViewController *viewController =
    [[ViewController alloc]
     initWithCollectionViewLayout:[self flowLayout]];

    self.window = [[UIWindow alloc] initWithFrame:
                   [[UIScreen mainScreen] bounds]];

    self.window.backgroundColor = [UIColor whiteColor];

    /* Задаем сборный вид в качестве корневого контроллера вида окна */
    self.window.rootViewController = viewController;
    [self.window makeKeyAndVisible];
    return YES;

}
```

Реализация контроллера сборного вида остается такой же, как в разделе 5.2. Если сейчас запустить приложение, то вы увидите просто черный экран, так как в стандартной реализации контроллера сборного вида фон вида даже не заменяется на белый. Пока нас это устраивает. Как минимум приложение уже не завершается аварийно, поскольку у нас уже есть объекты макета.

## См. также

Разделы 5.1 и 5.2.

# 5.4. Наполнение сборного вида простейшим содержимым

## Постановка задачи

Вы уже запрограммировали для вашего сборного вида макет с последовательной компоновкой, но пока не знаете, как отображать в нем ячейки.

## Решение

Для представления ваших ячеек либо напрямую воспользуйтесь классом `UICollectionViewCell`, либо произведите от него подкласс, на базе которого уже сможете написать собственную реализацию. Кроме того, как будет показано далее, у вас может быть файл `.xib`, ассоциированный с ячейкой.

## Обсуждение

В данном разделе я исхожу из того, что вы уже проработали разделы 5.1–5.3 и выполнили базовую настройку проекта.

Будем работать по порядку. Начнем с самого простого и самого быстрого способа создания ячеек. Инстанцируем объекты типа `UICollectionViewCell` и занесем их в сборный вид в нашем источнике данных. У класса `UICollectionViewCell` есть свойство вида с содержимым, называемое `contentView`, куда вы можете добавлять для отображения собственные виды. Кроме того, можете задавать и многие другие свойства ячейки, например цвет фона. Именно цветом фона мы и займемся в этом примере. Но перед тем, как начать, опишем, чего мы собираемся добиться в данном разделе, подробно объясним стоящие перед нами требования.

Мы собираемся запрограммировать сборный вид с последовательной компоновкой, в котором будут отображаться три секции. В каждой из секций будет находиться от 20 до 40 ячеек, причем в первой секции все ячейки красные, во второй — зеленые, в третьей — синие (рис. 5.5).

Итак, начнем. В контроллере сборного вида создадим метод, который может возвращать массив из трех цветов. Далее присвоим эти цвета ячейкам из каждой секции:

```
/* У нас будет три секции, и для каждой из них мы определим свой цвет ячеек. Для
представления цвета используются самые обычные экземпляры UIColor, которые мы позже
применим к каждой из ячеек в соответствующих секциях */
- (NSArray *) allSectionColors{

  static NSArray *allSectionColors = nil;

  if (allSectionColors == nil){
    allSectionColors = @[
                         [UIColor redColor],
                         [UIColor greenColor],
                         [UIColor blueColor],
                         ];
  }

  return allSectionColors;

}
```

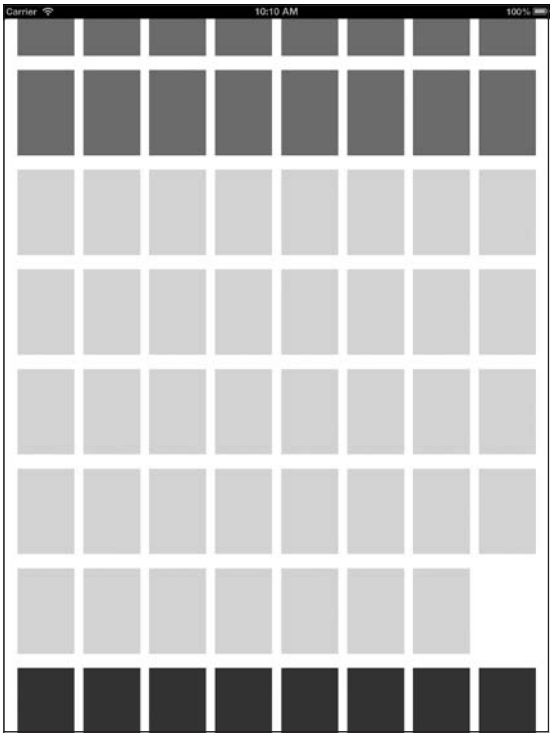


**Рис. 5.5.** Простой сборный вид с последовательной компоновкой, в котором отображаются три секции с ячейками разных цветов

После этого переопределите выделенный инициализатор `initWithCollectionViewLayout:` вашего контроллера сборного вида и зарегистрируйте `UICollectionViewCell` с конкретным идентификатором. Не волнуйтесь, если пока не совсем улавливаете ход мысли, взгляните на ситуацию таким образом: для каждой ячейки, которую должен отобразить ваш сборный вид, программа сначала просмотрит очередь ячеек, доступных для повторного использования, и определит, есть ли в этой очереди устраивающие ее ячейки. Если они найдутся, сборный вид извлечет такие ячейки из очереди, в противном случае он создаст новую ячейку и вернет ее вам для выполнения конфигурации.

В более ранних версиях iOS приходилось вручную создавать ячейки, если табличному виду не удавалось найти готовые ячейки для повторного использования (сборные виды в ранних версиях iOS отсутствовали). Однако с появлением новых API Apple выполнила очень интересную работу, связанную с многократно используемыми ячейками. Компания предоставила новые API как для табличных, так и для сборных видов. Поэтому вы можете зарегистрировать вызов и с табличным видом, и со сборным видом. Если же вам приходится конфигурировать новую ячейку, то вы просто требуете от табличного или сборного вида новую ячейку нужного рода. Если такая ячейка имеется в очереди многократного использования, то она (ячейка) будет вам предоставлена. Если нет — то табличный или сборный вид автоматически создаст такую ячейку. Этот механизм называется *регистрацией многоразовой ячейки*, он реализуется двумя способами:

- регистрацией ячейки с использованием имени класса;
- регистрацией ячейки с использованием `.xib`-файла.

Оба этих способа регистрации многоразовых ячеек вполне хороши и отлично работают со сборными видами. Чтобы зарегистрировать новую ячейку для сборного вида, воспользовавшись ее именем класса, применяется метод `registerClass:forCellWithReuseIdentifier:` класса `UICollectionView`, где идентификатор — обычная строка, которую вы сообщаете сборному виду. При попытке получить многоразовые ячейки вы запрашиваете у сборного вида ячейку с заданным идентификатором. Чтобы зарегистрировать со сборным видом `.xib`-файл, необходимо использовать метод экземпляра `registerNib:forCellWithReuseIdentifier:` сборного вида. Идентификатор этого метода также вполне функционален, об этом рассказано ранее в данном абзаце.

`Nib`-файл — это объект типа `UINib`, с ним мы подробнее познакомимся далее в этой главе.

```
- (instancetype) initWithCollectionViewLayout:(UICollectionViewLayout *)layout{

  self = [super initWithCollectionViewLayout:layout];
  if (self != nil){
    /* Регистрируем со сборным видом ячейку для ее удобного получения */
    [self.collectionView registerClass:[UICollectionViewCell class]
    forCellWithReuseIdentifier:kCollectionViewCellIdentifier];
  }
  return self;

}
```

Как видите, в качестве идентификатора для ячеек мы используем константное значение `kCollectionViewCellIdentifier`. Необходимо определить его в контроллере вида:

```
#import "ViewController.h"

static NSString *kCollectionViewCellIdentifier = @"Cells";

@implementation ViewController
```

В стандартной реализации сборного вида будет содержаться всего одна секция, если только вы не реализуете в источнике данных метод `numberOfSections-InCollectionView:`. В этом сборном виде мы хотим сделать три секции, так что реализуем его:

```
- (NSInteger)numberOfSectionsInCollectionView
           :(UICollectionView *)collectionView{
  return [self allSectionColors].count;
}
```

Одно из требований, предъявляемых к нашему приложению, такое: каждая секция должна содержать не менее 20, но не более 40 ячеек. Эту задачу можно решить с помощью функции `arc4random_uniform(x)`. Она возвращает положительные целые числа в диапазоне от 0 до *x*, где *x* — параметр, который вы сообщаете этой функции. Следовательно, если требуется сгенерировать число в диапазоне от 20 до 40, всего лишь нужно прибавить 20 к возвращаемому значению этой функции, а значение *x* также сделать равным 20. Зная это, реализуем метод `collectionView:numberOfItemsInSection:` из источника данных сборного вида:

```
- (NSInteger)collectionView:(UICollectionView *)collectionView
     numberOfItemsInSection:(NSInteger)section{
  /* Генерируем от 20 до 40 ячеек для заполнения каждой секции */
  return 20 + arc4random_uniform(21);
}
```

Наконец, требуется предоставить ячейки для сборного вида. Для этого реализуем метод `collectionView:cellForItemAtIndexPath:`, относящийся к источнику данных сборного вида:

```
- (UICollectionViewCell *)collectionView:(UICollectionView *)collectionView
                  cellForItemAtIndexPath:(NSIndexPath *)indexPath{

  UICollectionViewCell *cell =
  [collectionView
   dequeueReusableCellWithReuseIdentifier:kCollectionViewCellIdentifier
   forIndexPath:indexPath];

  cell.backgroundColor = [self allSectionColors][indexPath.section];

  return cell;

}
```

Индексные пути просто содержат номер секции и номер строки. Поэтому индексный путь 0, 1 указывает, что речь идет о второй строке первой секции, поскольку индексы имеют нулевое основание. Если же мы захотим указать пятую строку десятой секции, то обозначим индексный путь как 9, 4. Индексные пути очень широко используются при работе с табличными и сборными видами, так как органично подходят для описания секций, каждая из которых наполнена ячейками. Делегаты и источники данных для табличных и сборных видов при работе указывают целевую ячейку именно по ее индексному пути. Например, если пользователь нажмет ячейку в сборном виде, то вы получите его индексный путь. С помощью этого индексного пути вы также сможете просмотреть базовую структуру данных конкретной ячейки (речь идет о данных, которые использовались в вашем классе для создания этой ячейки).

Как видите, здесь используется метод экземпляра `dequeueReusableCellWithReuseIdentifier:forIndexPath:`, относящийся к сборному виду. Этот метод применяется для извлечения многоразовых ячеек из очереди. Этот метод ожидает получения двух параметров: идентификатора ячейки, которую вы ранее зарегистрировали с этим сборным видом, а также индексного пути, по которому должна быть отображена ячейка. Индексный путь вы получаете в том же самом методе `collectionView:cellForItemAtIndexPath:` в качестве параметра, поэтому остается всего лишь сообщить идентификатор ячейки.

Возвращаемое значение этого метода представляет собой ячейку типа `UICollectionViewCell`, которую можно сконфигурировать. В данной реализации нам придется сделать всего одну вещь: задать в качестве фонового цвета ячейки тот цвет, который мы ранее выбрали для всех ячеек данной секции.

Итак, остался последний шаг перед завершением данного примера. Сделаем фон сборного вида белым, чтобы он выглядел немного лучше, чем со стандартным черным фоном. Реализуйте метод `viewDidLoad` контроллера сборного вида и задайте фоновый цвет для данного вида прямо в этом методе:

```
- (void) viewDidLoad{
[super viewDidLoad];
self.collectionView.backgroundColor = [UIColor whiteColor];
}
```

Экземпляр UICollectionViewController имеет вид типа UIView, к которому можно получить доступ по его свойству view. Не путайте этот вид со свойством collectionView вашего контроллера, соответствующим той сущности, в которой располагается сам сборный вид.

Красота решения, предложенного в данном разделе, заключается в том, что оно отлично работает и на iPad, и на iPhone. На рис. 5.5 показано, как результат выглядит на iPad, на рис. 5.6 — как на iPhone.

## См. также

Разделы 5.1–5.3.

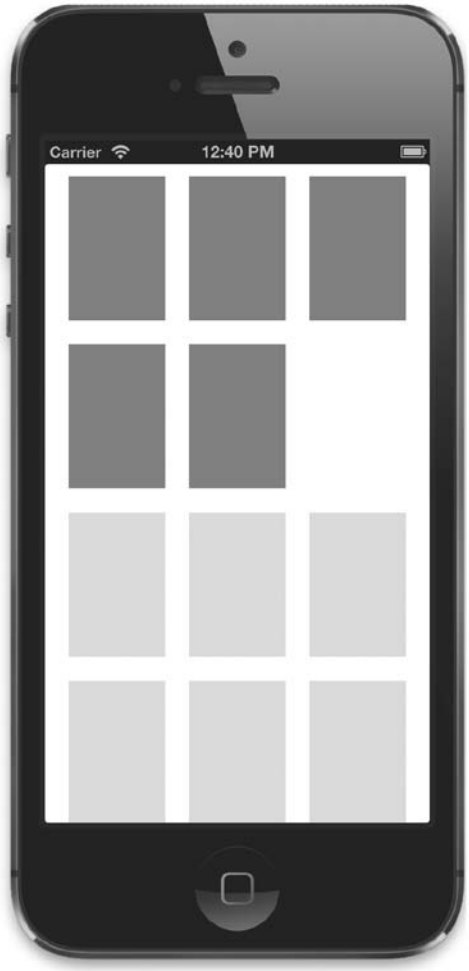


**Рис. 5.6.** Простой сборный вид, изображенный в эмуляторе iPhone

# 5.5. Заполнение сборных видов специальными ячейками с помощью XIB-файлов

## Постановка задачи

Требуется сконфигурировать ячейки сборного вида в конструкторе интерфейса и заполнить ими сборный вид для последующего отображения.

## Решение

Выполните следующие шаги.

1. Создайте подкласс `UICollectionViewCell` и назовите его (в данном примере мы будем использовать имя `CollectionViewCell`).
2. Создайте *пустой* `.xib`-файл и назовите его `MyCollectionViewCell.xib`.
3. Поместите в конструктор интерфейса ячейку сборного вида (ее вы найдете в библиотеке объектов). Она должна оказаться в вашем пустом `.xib`-файле (рис. 5.7). В конструкторе интерфейсов измените имя объекта-ячейки на `MyCollectionViewCell` (рис. 5.8). Поскольку вы устанавливаете такую ассоциацию, когда загружаете `.xib`-файл программно, специальный класс `MyCollectionViewCell` будет автоматически попадать в память. Волшебство, да и только!


**Рис. 5.7.** Объект пользовательского интерфейса «ячейка сборного вида» в библиотеке объектов конструктора интерфейса

4. Оформите ячейку в конструкторе интерфейса. Необходимо гарантировать, что для каждого компонента пользовательского интерфейса, который вы помещаете в ячейку, создается также ассоциированный с ней объект `IBOutlet`, расположенный либо в заголовочном файле, либо в файле реализации вашего класса (`MyCollectionViewCell`).
5. Зарегистрируйте с вашим сборным видом `.nib`-файл, воспользовавшись для этого методом экземпляра `registerNib:forCellWithReuseIdentifier:` сборного вида. Чтобы загрузить `.nib`-файл в память, используется метод класса `nibWithNibName:bundle:`, относящийся к классу `UINib`. Об этом методе мы вскоре поговорим.

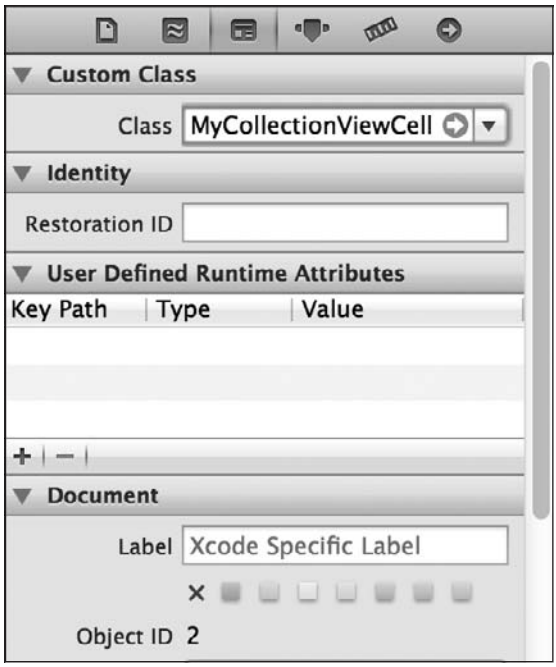


**Рис. 5.8.** Присваивание специального класса .xib-файлу ячейке специального сборного вида

## Обсуждение

Ранее в данном разделе вы узнали о том, что для специальной ячейки необходимо создать `.xib`-файл и назвать этот файл `MyCollectionViewCell.xib`. Не забывайте, что ваш `.xib`-файл может называться и совершенно иначе. Тем не менее ради простоты примера и для того, чтобы читатели видели во всей главе одно и то же соглашение об именованиях, будем пользоваться вышеупомянутым именем. Итак, продолжите и создайте пустой `.xib`-файл, выполнив следующие шаги.

1. Откройте File ▸ New ▸ File (Файл ▸ Новый ▸ Файл).
2. В левой части экрана в категории iOS выберите запись User Interface (Пользовательский интерфейс), а в правой части — вариант Empty (Пустой).
3. Теперь система запросит у вас семейство устройств, к которому относится данный `.xib`-файл. Здесь просто выберите iPhone.
4. Далее вам будет предложено сохранить `.xib`-файл на диск. Сохраните его под именем `MyCollectionViewCell.xib`.

Кроме того, потребуется создать класс, который вы сможете связать с содержимым `.xib`-файла. Этот класс мы назовем `MyCollectionViewCell`, он будет наследовать от `UICollectionViewCell`. Чтобы его создать, выполните следующие шаги.

1. Откройте File ▸ New ▸ File (Файл ▸ Новый ▸ Файл).
2. В диалоговом окне для создания нового файла в категории iOS выберите вариант Cocoa Touch. В правой части экрана выберите Objective-C class (Класс Objective-C).

3. Назовите класс `MyCollectionViewCell` и выберите `UICollectionViewCell` в качестве его класса-предка.
4. Когда вам будет предложено сохранить файл на диске, так и сделайте.

Теперь нужно ассоциировать созданный класс с `.xib`-файлом. Для этого выполните следующие шаги.

1. Откройте файл `MyCollectionViewCell.xib` в конструкторе интерфейса. В библиотеке объектов просто найдите объект Collection View Cell (Ячейка сборного вида) и поместите ее в `.xib`-файл. По умолчанию эта ячейка будет очень маленькой (50 × 50 точек) и будет иметь черный фон.
2. Явно выберите эту ячейку в вашем `.xib`-файле, щелкнув на ней. Откройте инспектор идентичности (Identity Inspector) в конструкторе интерфейса и измените значение поля Class (Класс) на `MyCollectionViewCell`, как было показано на рис. 5.8.

Далее следует добавить в ячейку нужные компоненты пользовательского интерфейса. Позже при заполнении сборного вида информацией значения этих компонентов можно будет изменить. Для данного примера лучше всего подойдет вид с изображением. Поэтому, когда у вас в конструкторе интерфейса открыт файл `MyCollectionViewCell.xib`, поместите в него экземпляр `UIImageView`. Подключите этот вид с изображением к заголовочному файлу вашей ячейки (`MyCollectionViewCell.h`) и назовите его `imageViewBackgroundImage`, чтобы заголовочный файл ячейки выглядел примерно так:

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

@interface MyCollectionViewCell : UICollectionViewCell

@property (weak, nonatomic) IBOutlet UIImageView *imageViewBackgroundImage;

@end
```

Мы собираемся заполнить этот вид разными изображениями. В этом разделе я создал для работы три простых изображения, каждое размером 50 × 50 точек. Вы можете пользоваться любыми другими картинками на ваш выбор — просто поищите их в Интернете. Когда найдете понравившиеся вам картинки, добавьте их в свой проект. Убедитесь, что изображения называются `1.png`, `2.png` и `3.png` и что их увеличенные вдвое аналоги для сетчатого дисплея называются `1@2x.png`, `2@2x.png` и `3@2x.png`.

В данном примере мы собираемся отобразить примерно такой же пользовательский интерфейс, как на рис. 5.5. Но мы будем задавать для ячеек не цвета, а фоновые изображения, причем случайные. Поэтому целесообразно использовать в качестве основы для данного примера тот код, который мы написали в разделе 5.4, так как результат будет очень похожим.

Первое изменение, которое предстоит внести, заключается в написании метода, с помощью которого мы будем возвращать случайное изображение. Как было объяснено ранее, у нас есть массив изображений. После инстанцирования массива нам понадобится удобный небольшой метод, который будет выбирать из массива случайное изображение:

```
- (NSArray *) allImages{

  static NSArray *AllSectionImages = nil;

  if (AllSectionImages == nil){
    AllSectionImages = @[
                         [UIImage imageNamed:@"1"],
                          [UIImage imageNamed:@"2"],
                          [UIImage imageNamed:@"3"]
                         ];
  }

  return AllSectionImages;

}

- (UIImage *) randomImage{
  return [self allImages][arc4random_uniform([self allImages].count)];
}
```

Далее потребуется переопределить выделенный метод-инициализатор контроллера сборного вида, чтобы зарегистрировать `.nib`-файл `MyCollectionViewCell` с этим сборным видом:

```
- (instancetype) initWithCollectionViewLayout:(UICollectionViewLayout *)layout{

  self = [super initWithCollectionViewLayout:layout];
  if (self != nil){
    /* Регистрируем nib-файл со сборным видом для удобства получения информации */
    UINib *nib = [UINib nibWithNibName:
                  NSStringFromClass([MyCollectionViewCell class])
                                bundle:[NSBundle mainBundle]];

    [self.collectionView registerNib:nib
          forCellWithReuseIdentifier:kCollectionViewCellIdentifier];
  }
  return self;
}
```

В ответ на запрос о том, сколько у нас секций, также возвратим случайное число в диапазоне от 3 до 6. Это требование не является обязательным — мы вполне могли бы обойтись и одной секцией, но если их будет больше, это точно не помешает. Кроме того, в каждой секции должно быть от 10 до 15 ячеек:

```
- (NSInteger)numberOfSectionsInCollectionView
           :(UICollectionView *)collectionView{
  /* От 3 до 6 секций */
  return 3 + arc4random_uniform(4);
}

- (NSInteger)collectionView:(UICollectionView *)collectionView
```

```
    numberOfItemsInSection:(NSInteger)section{
  /* В каждой секции - от 10 до 15 ячеек */
  return 10 + arc4random_uniform(6);
}
```

Наконец, запросим у сборного вида ячейки, а затем сконфигурируем их со случайными фоновыми изображениями:

```
- (UICollectionViewCell *)collectionView:(UICollectionView *)collectionView
  cellForItemAtIndexPath:(NSIndexPath *)indexPath{

    MyCollectionViewCell *cell =
    [collectionView
     dequeueReusableCellWithReuseIdentifier:kCollectionViewCellIdentifier
     forIndexPath:indexPath];

  cell.imageViewBackgroundImage.image = [self randomImage];
  cell.imageViewBackgroundImage.contentMode = UIViewContentModeScaleAspectFit;

  return cell;

}
```

Как видите, мы используем специальный класс MyCollectionViewCell в контроллере сборного вида. Чтобы программа успешно скомпилировалась, необходимо включить заголовочный файл ячейки в реализацию контроллера вида, вот так:

```
#import "ViewController.h"
#import "MyCollectionViewCell.h"

...
```

Запустив приложение, вы увидите примерно такую картинку, как на рис. 5.9. Разумеется, если вы используете в примере другие изображения, она будет другой, но у меня тут показаны нотки.

## См. также

Раздел 5.4.

# 5.6. Обработка событий в сборных видах

## Постановка задачи

Необходимо обрабатывать события, происходящие в сборных видах, например касания.

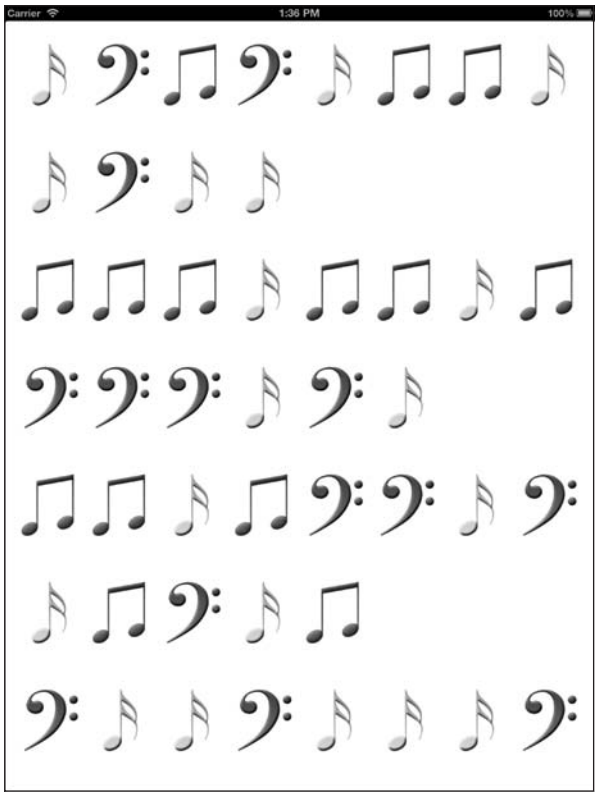


**Рис. 5.9.** Сборный вид со специальными ячейками, загруженными из .nib-файла

## Решение

Присвойте делегат сборному виду. В других случаях не придется делать даже этого. Порой достаточно просто слушать интересующие вас события в классах ячеек и обрабатывать их прямо в этих классах.

## Обсуждение

У сборных видов есть свойства `delegate`, которые должны соответствовать протоколу `UICollectionViewDelegate`. Делегатный объект, создаваемый с их помощью, будет получать различные делегатные вызовы от сборного вида, сообщающего делегату о различных событиях, например о том, что элемент был подсвечен или выделен. Необходимо различать подсвеченное и выделенное состояние ячейки сборного вида. Когда пользователь нажимает пальцем на ячейку сборного вида, но не поднимает палец сразу после этого, ячейка под пальцем становится *подсвеченной*. Когда пользователь нажимает на ячейку, а затем сразу поднимает палец (это означает, что он хочет совершить с ячейкой какое-то действие), данная ячейка становится *выделенной*.

Ячейки сборных видов, относящиеся к типу UICollectionViewCell, имеют два очень полезных свойства — highlighted и selected.

Если вы хотите всего лишь изменить визуальное оформление вашей ячейки, когда она выделена, то задача тем более упрощается, поскольку ячейки типа UICollectionViewCell предоставляют свойство selectedBackgroundView типа UIView. В качестве значения этого свойства можно задать любой валидный вид. Затем этот вид отобразится на экране, как только ячейка будет выделена. Продемонстрируем эти возможности на основе кода, который мы написали в разделе 5.5. Как вы помните, там мы создали специальную ячейку, одно из свойств которой (imageViewBackgroundImage) снабжало ее фоновым изображением. Изображение заполняло весь фон ячейки. В этот вид с изображением мы загружали специально подобранные картинки. Теперь мы собираемся залить *фон* ячейки голубым цветом, как только она будет выделена. Поскольку вид с изображением находится поверх всех остальных компонентов сборного вида, перед заданием фонового цвета нам придется гарантировать, что этот вид с изображением будет прозрачным. Для этого оттенок фона вида с изображением нужно изменить на проницаемый. Дело в том, что по умолчанию фон у вида с изображением непрозрачный, поэтому если расположить такой вид поверх другого вида, имеющего фоновый цвет, то этот фоновый цвет, естественно, виден не будет. Соответственно, чтобы оставался виден фоновый цвет того вида, который является вышестоящим для вида с изображением, фон самого вида с изображением должен быть прозрачным. Итак, начнем:

```
#import "MyCollectionViewCell.h"

@implementation MyCollectionViewCell

- (void) awakeFromNib{
  [super awakeFromNib];
  self.imageViewBackgroundImage.backgroundColor = [UIColor clearColor];
  self.selectedBackgroundView = [[UIView alloc] initWithFrame:self.bounds];
  self.selectedBackgroundView.backgroundColor = [UIColor blueColor];
}

@end
```

Вот и все! Теперь если нажать любую ячейку в вашей программе, она сразу приобретет голубой цвет фона.

Конечно, есть и другие операции, для выполнения которых требуется слушать различные события, происходящие в сборном виде. Например, может понадобиться воспроизвести звук или анимацию, как только оказывается выделенной ячейка. Допустим, когда пользователь прикасается к ячейке на экране, мы хотим задействовать следующую анимацию: немедленно скрыть ячейку, а потом снова ее отобразить. Эта анимация повторяется с очень высокой частотой, в результате чего ячейка постепенно вырисовывается или постепенно исчезает из виду. Если мы хотим добиться именно такого эффекта, для начала зададим делегат для нашего сборного вида, так как в описанном сценарии мы действительно будем получать от вида множество событий. Как было указано ранее, ваш делегатный объект должен соответствовать протоколу UICollectionViewDelegate. В этом протоколе есть несколько

полезных методов, которые мы можем реализовать. Далее перечислены некоторые важнейшие методы этого протокола.

Протокол UICollectionViewDelegateFlowLayout, как и рассмотренный нами в главе 4 протокол UITableViewDelegate, позволяет сообщать информацию о ваших элементах — например, значения их высоты и ширины, — а потом передавать эти значения макету с последовательной компоновкой. Можно сразу предоставить для всех элементов такого макета обобщенное значение размера, тогда они получатся одинаковыми. Другой вариант — реагировать на соответствующие сообщения, которые вы будете получать от протокола делегата макета с последовательной компоновкой. В этих сообщениях программа будет запрашивать у вас значения размеров для тех или иных ячеек в макете.

- `collectionView:didHighlightItemAtIndexPath:` — вызывается в делегате, когда ячейка подсвечивается.
- `collectionView:didUnhighlightItemAtIndexPath:` — вызывается в делегате, когда ячейка выходит из подсвеченного состояния. Этот метод срабатывает, когда пользователь успешно завершает событие касания (попадает пальцем по нужному элементу, а потом поднимает палец, совершая, таким образом, жест касания). В другом случае этот метод срабатывает, когда пользователь отменяет сделанное ранее выделение, выводя палец за пределы ячейки.
- `collectionView:didSelectItemAtIndexPath:` — этот метод вызывается в делегатном объекте, когда конкретная ячейка становится выделенной. Ячейка всегда является подсвеченной, перед тем как стать выделенной.
- `collectionView:didDeselectItemAtIndexPath:` — вызывается в делегате, когда ячейка выходит из выделенного состояния.

Итак, напишем приложение в соответствии с изложенными выше требованиями. Мы хотим, чтобы ячейка «развоплощалась», а потом вновь «вырисовывалась» на экране, когда ее выделяют. В экземпляре `UICollectionViewController` реализуем метод `collectionView:didSelectItemAtIndexPath:`, вот так:

```
#import "ViewController.h"
#import "MyCollectionViewCell.h"

static NSString *kCollectionViewCellIdentifier = @"Cells";

@implementation ViewController

- (void) collectionView:(UICollectionView *)collectionView
  didSelectItemAtIndexPath:(NSIndexPath *)indexPath{

    UICollectionViewCell *selectedCell =
        [collectionView cellForItemAtIndexPath:indexPath];

    const NSTimeInterval kAnimationDuration = 0.20;

    [UIView animateWithDuration:kAnimationDuration animations:^{
        selectedCell.alpha = 0.0f;
  } completion:^(BOOL finished) {
    [UIView animateWithDuration:kAnimationDuration animations:^{
```

```
        selectedCell.alpha = 1.0f;
    }];
  }];
}

...
```

Мы пишем этот код в контроллере сборного вида, который по умолчанию выбирается системой и в качестве источника данных, и в качестве делегата этого сборного вида. Он соответствует протоколам UICollectionViewDataSource и UICollectionViewDelegate. Следовательно, вы просто можете реализовать любой метод делегата или источника данных прямо в файле реализации вашего контроллера сборного вида.

В примере выше мы используем анимацию, но это не самое подходящее место, чтобы объяснять принципы работы анимации. Если вы хотите подробнее изучить, как в iOS создается простая анимация, обратитесь к главе 17 этой книги.

Итак, тут все было просто. Рассмотрим другой пример. Допустим, когда ячейка подсвечивается, мы хотим сделать ее вдвое крупнее обычного ее размера, а при выходе этой ячейки из подсвеченного состояния вернуть ей исходный размер. Таким образом, когда пользователь прикасается пальцем к ячейке (но еще не поднимает палец), ячейка увеличивается вдвое, а когда пользователь убирает палец — вновь уменьшается, тоже вдвое. Для этого нам потребуется реализовать в контроллере сборного вида методы `collectionView:didHighlightItemAtIndexPath:` и `collectionView:didUnhighlightItemAtIndexPath:` из протокола `UICollectionViewDelegate`. Как вы помните, контроллеры сборных видов по умолчанию соответствуют протоколам `UICollectionViewDelegate` и `UICollectionViewDataSource`:

```
#import "ViewController.h"
#import "MyCollectionViewCell.h"

static NSString *kCollectionViewCellIdentifier = @"Cells";
const NSTimeInterval kAnimationDuration = 0.20;

@implementation ViewController

- (void)       collectionView:(UICollectionView *)collectionView
  didHighlightItemAtIndexPath:(NSIndexPath *)indexPath{

    UICollectionViewCell *selectedCell =
      [collectionView cellForItemAtIndexPath:indexPath];

    [UIView animateWithDuration:kAnimationDuration animations:^{
      selectedCell.transform = CGAffineTransformMakeScale(2.0f, 2.0f);
    }];

  }

- (void) collectionView:(UICollectionView *)collectionView
  didUnhighlightItemAtIndexPath:(NSIndexPath *)indexPath{
```

```
    UICollectionViewCell *selectedCell =
      [collectionView cellForItemAtIndexPath:indexPath];

    [UIView animateWithDuration:kAnimationDuration animations:^{
      selectedCell.transform = CGAffineTransformMakeScale(1.0f, 1.0f);
    }];

  }

...
```

Как видите, мы используем функцию `CGAffineTransformMakeScale` из фреймворка Core Graphics для создания аффинного преобразования, а потом присваиваем это преобразование самой ячейке. Достигается нужный эффект: сначала ячейка увеличивается вдвое, а потом уменьшается до исходного размера. Эта функция подробнее описана в разделе 17.12.

## См. также

Разделы 5.2, 5.3, 5.5, 17.12.

# 5.7. Создание верхних и нижних колонтитулов в макете с последовательной компоновкой

## Постановка задачи

Требуется создать в сборном виде отдельные виды для верхнего и нижнего колонтитулов, так же как в табличном виде. При этом используется последовательная компоновка.

## Решение

Выполните следующие шаги.

1. Создайте по файлу `.xib` для верхнего и для нижнего колонтитулов.
2. Найдите в библиотеке объектов конструктора интерфейса по одному объекту Collection Reusable View и перетащите их в ваши `.xib`-файлы. Убедитесь, что эти многоразовые сборные виды являются единственными видами в своих `.xib`-файлах. Таким образом, многоразовый сборный вид становится корневым видом вашего `.xib`-файла. Именно так создаются колонтитулы в сборных видах.
3. Если вы хотите более полно контролировать поведение `.xib`-файлов, создайте класс Objective-C и ассоциируйте с ним корневой вид вашего `.xib`-файла. Таким образом, всякий раз, когда iOS будет загружать с диска содержимое `.xib`-файла, ассоциированный с ним класс также будет загружаться в память и вы будете получать доступ к иерархии видов в `.xib`-файле.

4. Инстанцируйте метод экземпляра `registerNib:forSupplementaryViewOfKind:withReuseIdentifier:` сборного вида и зарегистрируйте ваши `nib`-файлы для разновидностей видов `UICollectionElementKindSectionHeader` и `UICollectionElementKindSectionFooter`.
5. Чтобы правильно оформить виды верхних и нижних колонтитулов перед тем, как они будут отображены, реализуйте метод `collectionView:viewForSupplementaryElementOfKind:atIndexPath:` источника данных сборного вида, а в этом методе запустите другой метод сборного вида, `dequeueReusableSupplementaryViewOfKind:withReuseIdentifier:forIndexPath:`, чтобы извлечь из очереди многоразовый вид верхнего или нижнего колонтитула.
6. Наконец, необходимо убедиться, что размер видов для верхних и нижних колонтитулов задается путем присваивания соответствующих значений свойствам `headerReferenceSize` и `footerReferenceSize` макетного объекта, отвечающего за последовательную компоновку.

## Обсуждение

Итак, теперь нам требуется создать `.xib`-файлы для специальных верхних и нижних колонтитулов. Назовем их `Header.xib` и `Footer.xib`. Мы создаем их по тому же принципу, который описан в разделе 5.5, поэтому я не буду вновь повторять здесь этот материал. Убедитесь в том, что и для верхнего и для нижнего колонтитула у вас есть по одному классу Objective-C. Назовите их соответственно `Header` и `Footer`. Необходимо гарантировать, что оба этих класса наследуют от `UICollectionReusableView`. Закончив с этим, сконфигурируйте в конструкторе интерфейса подпись и кнопку, затем перетащите подпись в файл `Header`, а кнопку — в файл `Footer`. Свяжите их с вашими классами так, как показано на рис. 5.10 и 5.11.


**Рис. 5.10.** Конфигурирование ячейки верхнего колонтитула для сборного вида в конструкторе интерфейса


**Рис. 5.11.** Конфигурирование ячейки нижнего колонтитула для сборного вида в конструкторе интерфейса

Я связал подпись из верхнего колонтитула с классом `Header` с помощью свойства аутлета[1] в файле `Header.h`. Назовем аутлет просто `label`:

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

@interface Header : UICollectionReusableView
@property (weak, nonatomic) IBOutlet UILabel *label;
@end
```

То же самое я делаю и в нижнем колонтитуле, связав кнопку из файла `Footer.xib` с аутлетом из файла `Footer.h` и назвав аутлет `button`:

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

@interface Footer : UICollectionReusableView
@property (weak, nonatomic) IBOutlet UIButton *button;
@end
```

Теперь в контроллере сборного вида определим идентификаторы для ячеек верхнего и нижнего колонтитулов:

```
#import "ViewController.h"
#import "MyCollectionViewCell.h"
#import "Header.h"
#import "Footer.h"

static NSString *kCollectionViewCellIdentifier = @"Cells";
static NSString *kCollectionViewHeaderIdentifier = @"Headers";
static NSString *kCollectionViewFooterIdentifier = @"Footers";
```

[1] О том, что такое аутлет и чем такая связь отличается от action, подробно рассказано в статье по адресу http://habrahabr.ru/post/30553/. — *Примеч. пер.*

```
@implementation ViewController

...
```

Далее в методе-инициализаторе сборного вида зарегистрируем ячейку сборного вида, ячейку верхнего колонтитула и ячейку нижнего колонтитула. Для этого воспользуемся nib-файлами, которые мы загружаем в память:

```
- (instancetype) initWithCollectionViewLayout:(UICollectionViewLayout *)layout{

  self = [super initWithCollectionViewLayout:layout];
  if (self != nil){
    /* Регистрируем nib-файл со сборным видом для удобного получения */
    UINib *nib = [UINib nibWithNibName:
                  NSStringFromClass([MyCollectionViewCell class])
                        bundle:[NSBundle mainBundle]];

  [self.collectionView registerNib:nib
        forCellWithReuseIdentifier:kCollectionViewCellIdentifier];

  /* Регистрируем nib-файл верхнего колонтитула */
  UINib *headerNib = [UINib
                      nibWithNibName:NSStringFromClass([Header class])
                      bundle:[NSBundle mainBundle]];
  [self.collectionView registerNib:headerNib
        forSupplementaryViewOfKind:UICollectionElementKindSectionHeader
               withReuseIdentifier:kCollectionViewHeaderIdentifier];

  /* Регистрируем nib-файл нижнего колонтитула */
  UINib *footerNib = [UINib
                      nibWithNibName:NSStringFromClass([Footer class])
                      bundle:[NSBundle mainBundle]];
  [self.collectionView registerNib:footerNib
        forSupplementaryViewOfKind:UICollectionElementKindSectionFooter
               withReuseIdentifier:kCollectionViewFooterIdentifier];
  }
  return self;

}
```

Переходим к реализации метода `collectionView:viewForSupplemen taryEle-mentOfKind:atIndexPath:` сборного вида. Этот метод нужен нам для конфигурирования верхних и нижних колонтитулов и предоставления их обратно сборному виду:

```
- (UICollectionReusableView *)collectionView:(UICollectionView *)collectionView
          viewForSupplementaryElementOfKind:(NSString *)kind
                                atIndexPath:(NSIndexPath *)indexPath{

  NSString *reuseIdentifier = kCollectionViewHeaderIdentifier;
  if ([kind isEqualToString:UICollectionElementKindSectionFooter]){
    reuseIdentifier = kCollectionViewFooterIdentifier;
```

```
    }

    UICollectionReusableView *view =
    [collectionView dequeueReusableSupplementaryViewOfKind:kind
                                       withReuseIdentifier:reuseIdentifier
                                              forIndexPath:indexPath];

  if ([kind isEqualToString:UICollectionElementKindSectionHeader]){
    Header *header = (Header *)view;
    header.label.text = [NSString stringWithFormat:@"Section Header %lu",
    (unsigned long)indexPath.section + 1];
  }
  else if ([kind isEqualToString:UICollectionElementKindSectionFooter]){
    Footer *footer = (Footer *)view;

    NSString *title = [NSString stringWithFormat:@"Section Footer %lu",
                       (unsigned long)indexPath.section + 1];
    [footer.button setTitle:title forState:UIControlStateNormal];

  }

  return view;

}
```

Наконец, необходимо убедиться, что макету с последовательной компоновкой известно о том, что в сборном виде есть ячейки верхнего и нижнего колонтитулов. На основе кода, написанного в разделе 5.3, изменим метод `flowLayout` делегата нашего приложения следующим образом:

```
- (UICollectionViewFlowLayout *) flowLayout{

  UICollectionViewFlowLayout *flowLayout =
  [[UICollectionViewFlowLayout alloc] init];

  flowLayout.minimumLineSpacing = 20.0f;
  flowLayout.minimumInteritemSpacing = 10.0f;
  flowLayout.itemSize = CGSizeMake(80.0f, 120.0f);
  flowLayout.scrollDirection = UICollectionViewScrollDirectionVertical;
  flowLayout.sectionInset = UIEdgeInsetsMake(10.0f, 20.0f, 10.0f, 20.0f);
  /* Задаем базовый размер для видов с верхними и нижними колонтитулами */
  flowLayout.headerReferenceSize = CGSizeMake(300.0f, 50.0f);
  flowLayout.footerReferenceSize = CGSizeMake(300.0f, 50.0f);

  return flowLayout;

}
```

Итак, все готово! Если вы теперь запустите приложение в эмуляторе iPad, то увидите примерно такой результат, как на рис. 5.12.

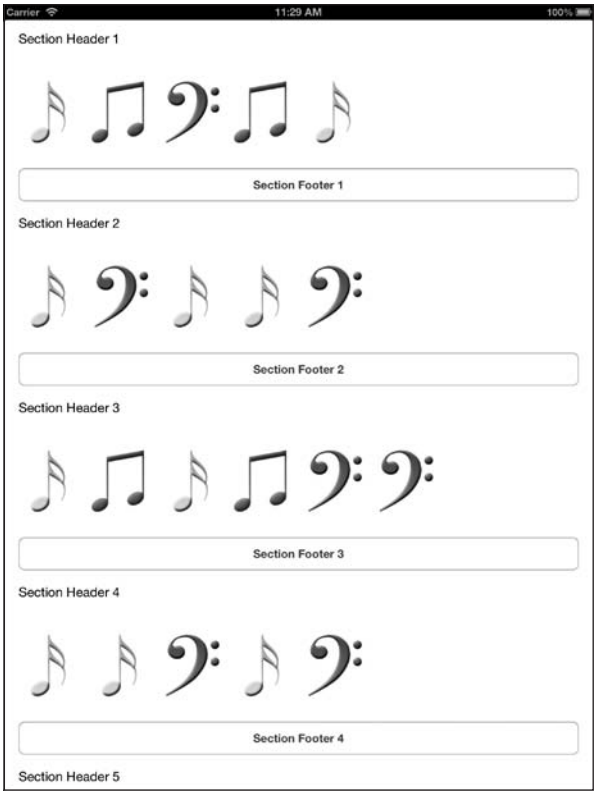


**Рис. 5.12.** Верхние и нижние колонтитулы в сборном виде

## См. также

Разделы 5.2, 5.3, 5.5.

# 5.8. Добавление собственных вариантов взаимодействий к сборным видам

## Постановка задачи

Вы хотели бы добавить к сборному виду собственные механизмы распознавания жестов, таких как щипок, чтобы реализовать собственные варианты поведений на базе уже имеющихся.

## Решение

Инстанцируйте механизм распознавания жестов, а потом просмотрите распознаватели жестов, уже имеющиеся в сборном виде, и проверьте, нет ли среди них такого,

который похож на нужный вам. Если найдете такой механизм, вызовите в нем метод `requireGestureRecognizerToFail:` и передайте этому методу в качестве параметра ваш собственный распознаватель жестов. Так вы гарантируете, что распознаватель жестов, имеющийся в сборном виде и напоминающий тот, что нужен вам, будет в случае необходимости подхватывать обработку жестов. Это станет происходить, если вашему распознавателю жестов не удастся обработать те или иные данные либо данные не будут соответствовать его требованиям/критериям. Таким образом, если ваш распознаватель способен обработать жест, он это сделает, в противном случае жест будет передан механизму распознавания, уже имеющемуся в сборном виде. Этот механизм обработает его сам.

Как только выполните описанную работу, добавьте ваш распознаватель жестов к сборному виду. Как вы помните, в экземпляре `UICollectionViewController` объект вашего сборного вида будет доступен через свойство контроллера `collectionView`, а *не* через свойство `view`.

## Обсуждение

В API iOS уже предусмотрено несколько механизмов обработки жестов, применяемых в сборных видах. Поэтому для добавления собственных распознавателей жестов к уже имеющейся коллекции сначала нужно убедиться в том, что ваши распознаватели жестов не будут функционально пересекаться с уже имеющимися. Для этого сначала нужно инстанцировать ваши собственные распознаватели жестов, а потом, как описано ранее, просмотреть массив таких распознавателей, уже имеющихся у сборного вида. Затем понадобится вызвать метод `requireGestureRecognizerToFail:` в классе распознавателя жестов того же типа, что и наш распознаватель жестов, который мы собираемся добавить к сборному виду.

Рассмотрим пример. В этом примере мы собираемся добавить к сборному виду возможность уменьшения и увеличения изображения (то есть его масштабирования). Этот пример будет выстроен на основе того, который мы подготовили в разделе 5.5. Итак, первым делом мы должны добавить распознаватель щипков в коллекцию распознавателей жестов, имеющихся в сборном виде. Мы сделаем это в методе `viewDidLoad` контроллера сборного вида:

```
- (void) viewDidLoad{
  [super viewDidLoad];
  self.collectionView.backgroundColor = [UIColor whiteColor];

  UIPinchGestureRecognizer *pinch = [[UIPinchGestureRecognizer alloc]
                                     initWithTarget:self
                                     action:@selector(handlePinches:)];
  for (UIGestureRecognizer *recognizer in
  self.collectionView.gestureRecognizers){
    if ([recognizer isKindOfClass:[pinch class]]){
      [recognizer requireGestureRecognizerToFail:pinch];
    }
  }
```

```
  [self.collectionView addGestureRecognizer:pinch];
}
```

Настраиваем распознаватель жестов щипка для вызова метода `handlePinches:` контроллера вида. Сейчас мы напишем этот метод:

```
- (void) handlePinches:(UIPinchGestureRecognizer *)paramSender{

  CGSize DefaultLayoutItemSize = CGSizeMake(80.0f, 120.0f);

  UICollectionViewFlowLayout *layout =
    (UICollectionViewFlowLayout *)self.collectionView.collectionViewLayout;

  layout.itemSize =
  CGSizeMake(DefaultLayoutItemSize.width * paramSender.scale,
             DefaultLayoutItemSize.height * paramSender.scale);

  [layout invalidateLayout];
}
```

В этом коде есть две очень важные детали.

1. Предполагается, что по умолчанию размер элемента в макете последовательной компоновки сборного вида имеет ширину 80 точек и высоту 120 точек. Именно так мы создали макет с последовательной компоновкой для сборного вида в разделе 5.3. Затем мы берем коэффициент масштабирования, полученный от распознавателя жестов щипка, и умножаем на него размер элементов из сборного вида. В результате эти экранные элементы могут уменьшиться или увеличиться в зависимости от того, как именно пользователь масштабирует экран.
2. После того как был изменен размер элемента, применяемый по умолчанию в макете с последовательной компоновкой, макет необходимо обновить. В табличных видах мы обновляли либо нужные секции таблицы, либо всю таблицу, но в данном случае обновляем или упраздняем макет, прикрепленный к сборному виду. Это делается, чтобы сборный вид полностью «перерисовал» себя после изменения макета. Поскольку сборный вид в каждый момент времени может содержать всего один макетный объект, при упразднении такого макетного объекта потребуется перезагрузить весь сборный вид. Если бы мы могли иметь отдельный макет для каждой секции, то могли бы перезагружать только те секции, которые связаны с данным макетом. Но, имея такой код, как сейчас, при упразднении макетного объекта придется перерисовывать весь сборный вид.

Теперь, запустив код, вы заметите, что можете взаимодействовать с экраном с помощью двух пальцев. Если вы сводите пальцы, то элементы вашего сборного вида увеличиваются в размере, а если разводите — уменьшаются.

## См. также

Разделы 5.3 и 5.5.

# 5.9. Представление контекстных меню в ячейках сборных видов

## Постановка задачи

Если пользователь нажимает на один из экранных элементов в вашем сборном виде и удерживает на нем палец, требуется вывести контекстное меню. С помощью команд из этого меню элемент можно будет скопировать, переместить и т. д.

## Решение

Контекстные меню по умолчанию встроены в сборные виды. Чтобы активизировать их, потребуется всего лишь реализовать следующие методы из протокола `UICollectionViewDelegate`.

- `collectionView:shouldShowMenuForItemAtIndexPath:` — среда времени исполнения передает этому методу индексный путь к элементу. Метод возвращает логическое значение, указывающее дальнейшее действие: должен этот элемент открывать контекстное меню или нет.
- `collectionView:canPerformAction:forItemAtIndexPath:withSender:` — среда времени исполнения передает этому методу селектор типа `SEL`. Можно проверить селектор (обычно для этого он преобразуется в строку, которая затем сравнивается со строкой, представляющей действие) и определить, хотите ли вы, чтобы указанное действие произошло. Возвратите `YES`, чтобы разрешить такое действие, либо `NO`, чтобы подавить его. Не забывайте, что вы всегда можете преобразовать селектор в строку, воспользовавшись методом `NSStringFromSelector`. Типичные примеры селекторов — `copy:` или `paste:` для команд контекстного меню Копировать или Вставить соответственно.
- `collectionView:performAction:forItemAtIndexPath:withSender:` — здесь выполняется действие, которое было с вашего разрешения отображено в сборном виде с помощью вышеупомянутых делегатных методов.

## Обсуждение

Не откладывая в долгий ящик, расширим код, написанный в разделе 5.5. Мы будем выводить в ячейках контекстное меню Copy (Копировать), если пользователь нажмет на ячейку и на некоторое время задержит на ней палец. Когда пользователь выбирает элемент в меню копирования, мы скопируем изображение из ячейки в буфер обмена. После этого пользователь сможет вставить это изображение в файлы из других программ, например из почтового приложения Mail.

Первым делом реализуем в делегате сборного вида метод `collectionView:shouldShowMenuForItemAtIndexPath:`. В данном примере мы работаем с контроллером сборного вида, который сам является и делегатом и источником данных. Поэтому фактически нам придется всего лишь реализовать вышеупомянутый метод в контроллере сборного вида, вот так:

```
- (BOOL) collectionView:(UICollectionView *)collectionView
  shouldShowMenuForItemAtIndexPath:(NSIndexPath *)indexPath{
    return YES;
}
```

Теперь мы хотим обеспечить, чтобы в ячейках нашего сборного вида отображалось лишь контекстное меню для копирования. В рамках этого примера рассмотрим, как можно отфильтровать доступные элементы меню и отобразить только нужные:

```
- (BOOL) collectionView:(UICollectionView *)collectionView
       canPerformAction:(SEL)action
     forItemAtIndexPath:(NSIndexPath *)indexPath
             withSender:(id)sender{

  if (action == @selector(copy:)){
    return YES;
  }

  return NO;
}
```

Как видите, нам даже не требуется преобразовывать селектор в строку, чтобы сравнить его с такими строками, как `copy:`. Мы всего лишь используем оператор равенства, чтобы проверить, отвечает ли запрошенный селектор нашим ожиданиям. Если это так, возвращаем `YES`, в противном случае — `NO`.

Наконец, нам потребуется реализовать в делегате метод `collectionView:performAction:forItemAtIndexPath:withSender:`. С помощью этого метода мы узнаем, было ли вызвано действие копирования, а потом копируем изображение, взятое из ячейки, в буфер обмена. Пользователь сможет вставить из буфера изображение в файл из совершенно другого приложения:

```
- (void) collectionView:(UICollectionView *)collectionView
          performAction:(SEL)action
     forItemAtIndexPath:(NSIndexPath *)indexPath
             withSender:(id)sender{

  if (action == @selector(copy:)){

    MyCollectionViewCell *cell = (MyCollectionViewCell *)[collectionView
                                  cellForItemAtIndexPath:indexPath];

    [[UIPasteboard generalPasteboard]
     setImage:cell.imageViewBackgroundImage.image];

  }

}
```

Теперь, если вы запустите приложение и нажмете один из элементов в сборном виде, а потом будете удерживать на нем палец, то получите примерно такой результат, как на рис. 5.13.

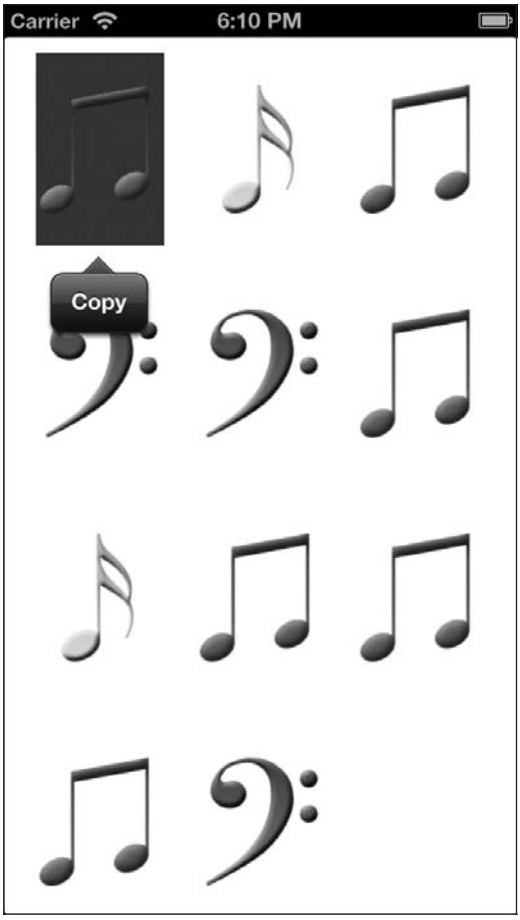


**Рис. 5.13.** Элемент контекстного меню, отображаемый в ячейке сборного вида

## См. также

Раздел 5.5.

# 6 Раскадровки

## 6.0. Введение

Программисты iOS уже привыкли работать с контроллерами видов. Мы умеем пользоваться навигационными контроллерами, чтобы выводить на экран и убирать с него контроллеры видов. Но Apple полагает, что такие задачи можно решать и проще, и поэтому в системе появились раскадровки. *Раскадровки* — это новый способ определения связей между экранами вашего приложения. Например, если в вашем приложении 20 уникальных контроллеров видов, вы написали эти контроллеры год назад, а *сейчас* снова изучаете исходный код, то вам придется снова распутывать все замысловатые соединения между контроллерами видов. Вы будете пытаться запомнить, какой именно контроллер вида поднимается вверх по стеку, когда пользователь совершает то или иное действие. Это может быть очень сложно, особенно если вы не слишком подробно документировали код. И вот тут вам поможет раскадровка. Раскадровка позволяет просматривать или создавать сразу весь пользовательский интерфейс приложения, а также выстраивать связи между контроллерами видов на одном экране. Да, все настолько просто.

Чтобы воспользоваться преимуществами, которые дает раскадровка, необходимо вплотную познакомиться с конструктором интерфейсов. Не волнуйтесь: обо всем важном рассказано в этой главе.

При работе с раскадровками каждый экран, наполненный значимым содержимым, называется *сценой*. Отношение между сценой и раскадровкой в iPhone можно сравнить с отношением вида к контроллеру вида. Весь контент сцены отображается на экране одновременно, соответственно, и пользователь воспринимает эту информацию одновременно. На iPad пользователь одновременно может просматривать более одной сцены, так как у планшета довольно большой экран.

При раскадровке возможен переход от одной сцены к другой. Применяемый в раскадровке процесс, в ходе которого один контроллер вида ставится выше другого, называется *сегвеем*. Еще одним примером перехода является ситуация с модальным контроллером вида, который на время поднимает сцену «снизу» экрана так, чтобы она заполнила весь экран. На iPad модальные окна обычно появляются в центре экрана, в это время остальная часть экрана затемняется. Таким образом подчеркивается, что в момент отображения модального окна именно оно является основным каналом ввода.

# 6.1. Добавление в раскадровку навигационного контроллера

## Постановка задачи

Требуется возможность управлять несколькими котроллерами видов в приложении, построенном на основе раскадровки.

## Решение

Задайте навигационный контроллер как исходный контроллер вида в файле раскадровки.

## Обсуждение

Если вы создали в Xcode новое универсальное приложение, воспользовавшись шаблоном Single View Application (Приложение с единственным видом), то у вас будет два файла раскадровок: `Main_iPhone.storyboard` и `Main_iPad.storyboard`. Если просмотреть их в конструкторе интерфейса, то легко заметить, что контроллер вида применяется в них в качестве корневого контроллера. На рис. 6.1 показано содержимое простого готового файла раскадровки для iPhone.

Чтобы заменить корневой контроллер вида в файле раскадровки на навигационный контроллер, выполните следующие шаги.

1. Выберите контроллер вида на холсте раскадровки.
2. В меню Edit (Правка) выберите команду Embed in (Встроить), а затем Navigation Controller (Навигационный контроллер) (рис. 6.2).

Как только справитесь с этим, вы заметите, что контроллер вида в раскадровке превратился в навигационный контроллер (рис. 6.3).

## См. также

Раздел 6.0.

# 6.2. Передача данных с одного экрана на другой

## Постановка задачи

Необходимо передавать данные из одной сцены в другую, используя раскадровку.

## Решение

Воспользуйтесь сегвеями, обеспечивающими плавные переходы.

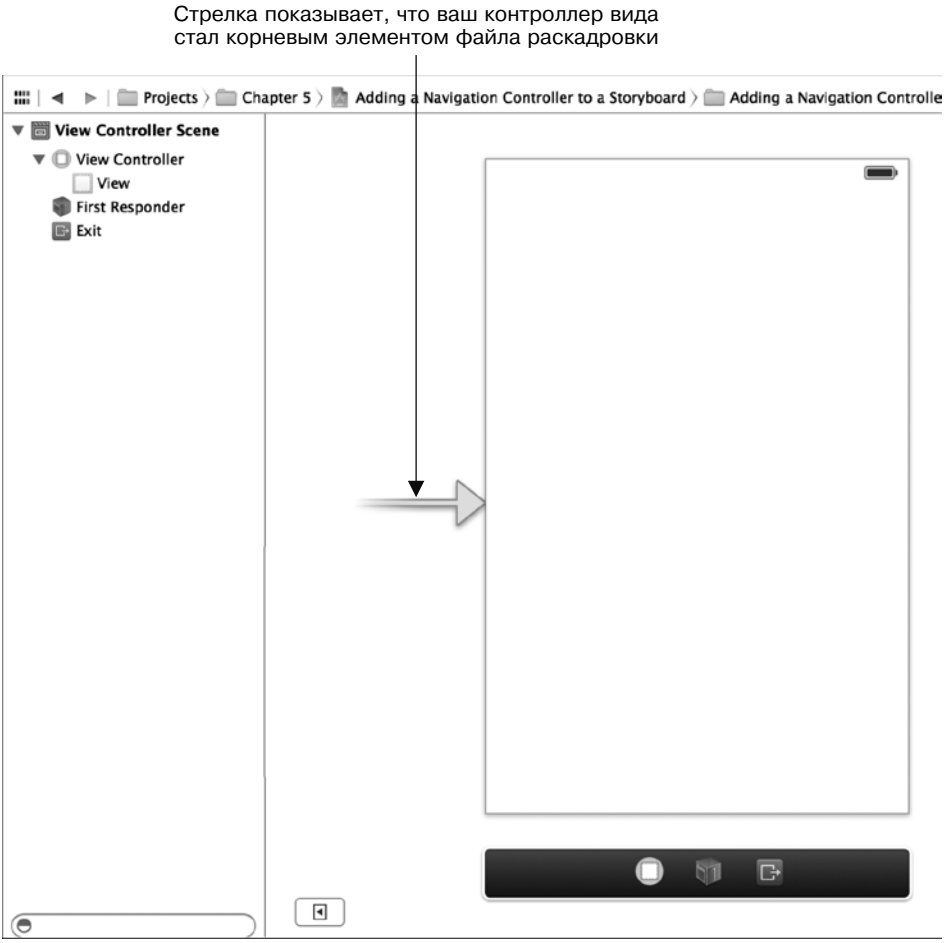


**Рис. 6.1.** Контроллер вида в качестве корневого элемента файла раскадровки

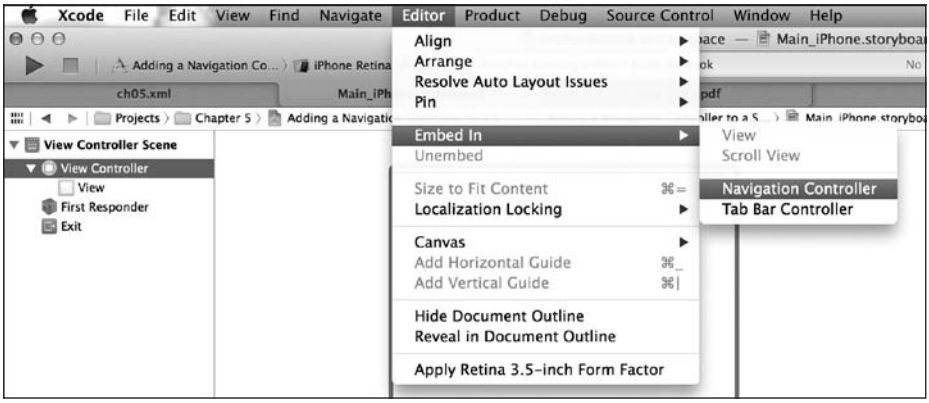


**Рис. 6.2.** Активизация контроллера вида в навигационном контроллере

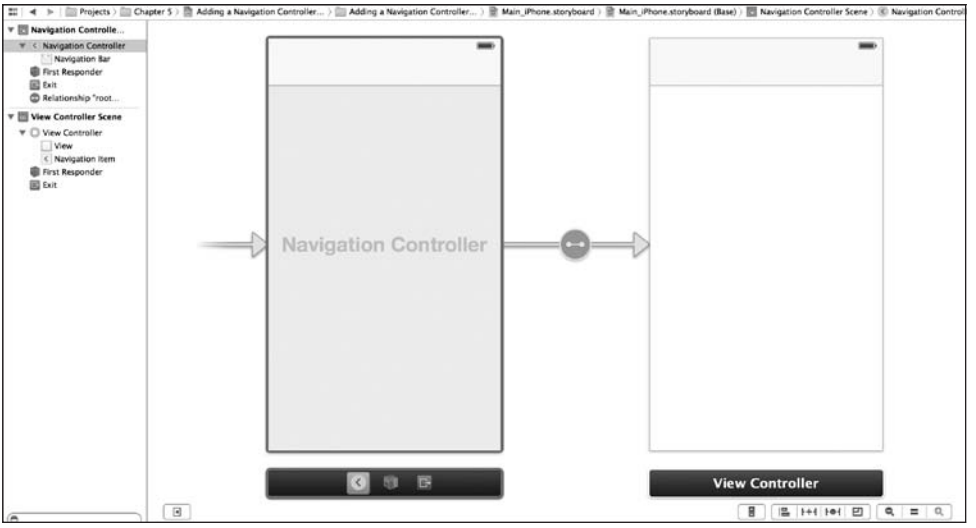


**Рис. 6.3.** Навигационный контроллер теперь является корневым контроллером раскадровки

## Обсуждение

*Сегвей* — это объект, напоминающий любые другие объекты языка Objective-C. Чтобы выполнить переход от одной сцены к другой, среда времени исполнения раскадровки создает объект-сегвей именно для этой цели[1]. Сегвей — это экземпляр класса `UIStoryboardSegue`. Чтобы начался переход, текущий контроллер вида (этот вид уходит с экрана по завершении плавного перехода) получает сообщение `prepareForSegue:sender:`, где в качестве параметра `prepareForSegue` будет использован объект типа `UIStoryboardSegue`. Если вы хотите передать какие-либо данные от актуального контроллера вида к контроллеру того вида, который вот-вот появится на экране, это нужно делать в методе `prepareForSegue:sender:`.

Для полноценной работы с этим разделом нужно выполнить инструкции из раздела 6.1, где в раскадровке создаются два контроллера видов внутри навигационного контроллера.

Рассмотрим прикладной пример с использованием сегвеев. В этом разделе мы собираемся отобразить на экране примерно такой контроллер вида, как показан на рис. 6.4.

Та информация, которую пользователь внесет в текстовое поле, будет передана второму контроллеру вида посредством сегвея и задана в качестве заголовка этого контроллера вида. Холст второго контроллера вида будет пуст. Итак, воспользуйтесь приемами, изученными в разделе 6.1, и поместите первый контроллер вида в навигационный контроллер. Теперь возьмите в библиотеке объектов другой

[1] Интересная статья о сегвеях и их роли в раскадровках: http://www.raywenderlich.com/ru/24949/. — *Примеч. пер.*

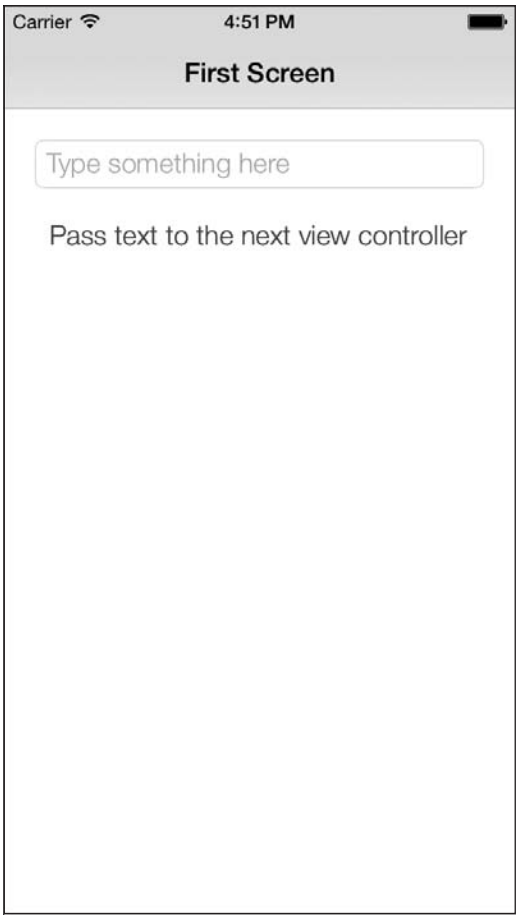


**Рис. 6.4.** Первый контроллер вида в нашем приложении; на контроллере вида есть текстовое поле и кнопка

контроллер вида, поместите его в раскадровку, а также разместите в первом контроллере вида кнопку и текстовое поле. Вы заметите, что положение текстового поля и кнопки получается примерно таким, как на рис. 6.4, но такое сходство не является обязательным. Можете расположить элементы как хотите. Теперь, удерживая нажатой клавишу Ctrl, наведите указатель на экранную кнопку и нажмите и не отпускайте кнопку мыши. На экране появится линия. Перетащите ее на второй контроллер вида (рис. 6.5). Откроется диалоговое окно, в нем выберите элемент Push. Сделав это, вы устанавливаете связь между кнопкой и вторым контроллером вида. Когда кнопка нажимается, контроллер вида оказывается на верхней позиции в стеке навигационного контроллера.

В конструкторе интерфейса видно, что мы создали сегвей между первым и вторым контроллерами вида. Щелкните на сегвее в инспекторе атрибутов (Attribute Inspector), присвойте ему идентификатор `pushSecondViewController` (рис. 6.6).

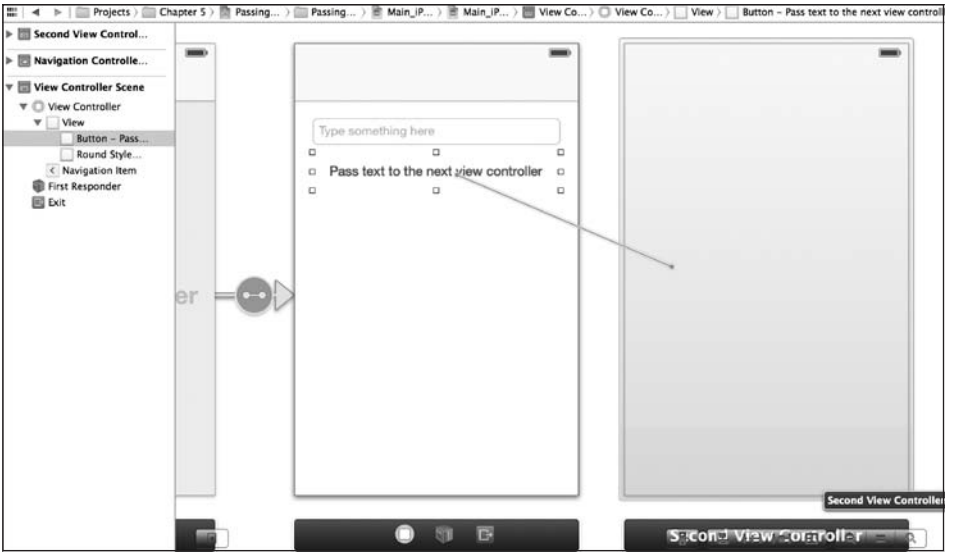


**Рис. 6.5.** Создание связи между кнопкой и вторым контроллером вида; связь срабатывает при нажатии кнопки

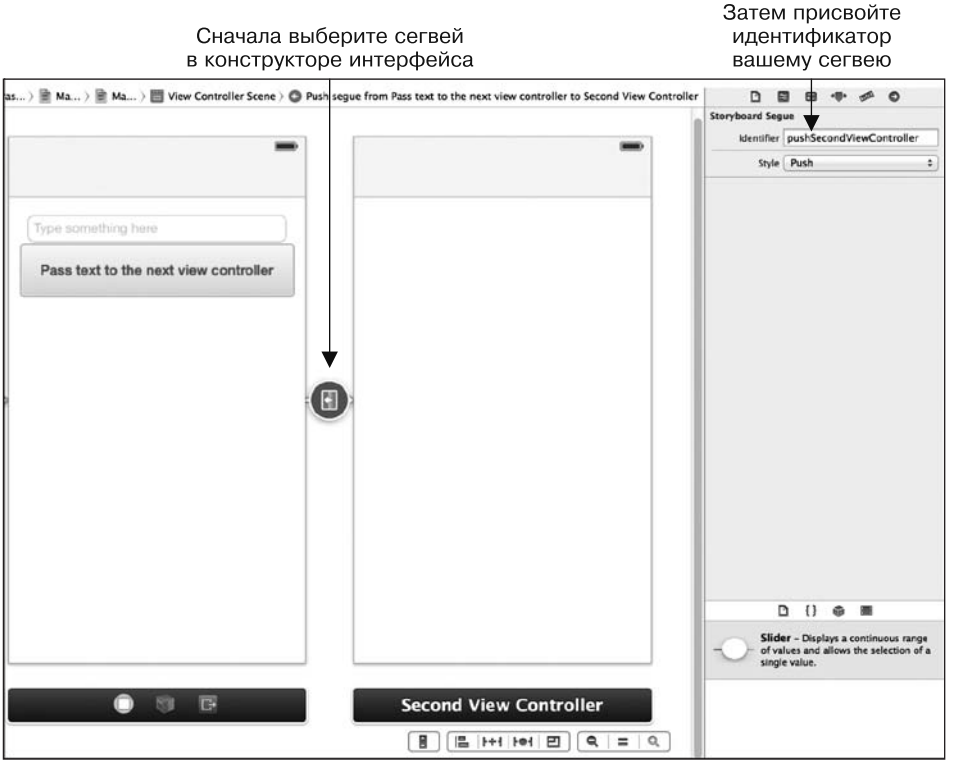


**Рис. 6.6.** Присваивание идентификатора сегвею

Может возникнуть вопрос: а зачем вообще нужен этот идентификатор? Дело в том, что мы реализуем специальный метод контроллера вида, который сначала будет спрашивать, допустимо ли сейчас выполнить такой сегвей. В этом методе мы проверим текст, находящийся в текстовом поле, и, если это поле окажется пустым, не позволим пользователю перейти на следующий экран. Метод, который будет вызываться в контроллере вида, называется `shouldPerformSegueWithIdentifier:sender:`, он относится к классу `UIViewController`. Вы можете использовать значение типа `NSString`, записываемое в его параметр `shouldPerformSegueWithIdentifier`, чтобы получить идентификатор того сегвея, который собирается выполнить система. После этого вы будете должны возвратить значение `YES`, если планируемый сегвей вас устраивает, и `NO` — в противном случае. Если вернуть `NO`, то сегвей с заданным идентификатором выполнен не будет. Но блокировать переход, никак не дав знать об этом пользователю, — безусловно, порочная практика. Поэтому, если поле оказывается пустым, а пользователь пытается нажать кнопку и перейти на следующий экран, мы отобразим для него такое диалоговое окно, как на рис. 6.7.


**Рис. 6.7.** Пользователь не сможет перейти на следующий экран, пока не введет текст в следующее поле

Итак, наконец перейдем к реализации первого контроллера вида. Предполагаю, что вы уже соединили текстовое поле с контроллером вида (поле выступает в качестве аутлета для этого контроллера) и можете получить доступ к его свойству text, перед тем как произойдет сегвей:

```
#import "ViewController.h"
#import "SecondViewController.h"

@interface ViewController () <UITextFieldDelegate>
@property (weak, nonatomic) IBOutlet UITextField *textField;
@end

@implementation ViewController

- (void) viewDidLoad{
    [super viewDidLoad];
    self.title = @"First Screen";
  }

- (BOOL) textFieldShouldReturn:(UITextField *)textField{
  [textField resignFirstResponder];
  return YES;
}

- (void) displayTextIsRequired{

  UIAlertView *alert = [[UIAlertView alloc]
                        initWithTitle:nil
                        message:@"Please enter some text in the text field"
                        delegate:nil
                        cancelButtonTitle:nil
                        otherButtonTitles:@"OK", nil];
  [alert show];

}

- (BOOL) shouldPerformSegueWithIdentifier:(NSString *)identifier
  sender:(id)sender{
    /* Проверяем, есть ли в текстовом поле какой-либо текст. Если текста
    там нет, то отображаем пользователю соответствующее сообщение
    и не позволяем перейти на следующий экран */
    if ([identifier isEqualToString:@"pushSecondViewController"]){

      if ([self.textField.text length] == 0){
          [self displayTextIsRequired];
          return NO;
    }
  };

  return YES;
```

```
}
- (void) prepareForSegue:(UIStoryboardSegue *)segue sender:(id)sender{

  if ([segue.identifier isEqualToString:@"pushSecondViewController"]){
      SecondViewController *nextController =
      segue.destinationViewController;
      [nextController setText:self.textField.text];

  }

}

@end
```

Метод `prepareForSegue:sender:` этого контроллера вида вызывает метод экземпляра `setText:`, относящийся к классу `SecondViewController`. Как понятно из названия, это класс второго контроллера вида. Мы просто реализуем этот метод следующим образом:

```
#import "SecondViewController.h"

@interface SecondViewController ()
@end

@implementation SecondViewController

- (void) setText:(NSString *)paramText{
  self.title = paramText;
}

@end
```

Вот и все. Теперь, если вы запустите предложение и введете в текстовое поле текст, скажем, `Hello, World!`, а потом нажмете кнопку, то увидите примерно такую картинку, как на рис. 6.8.

## См. также

Раздел 6.1.

# 6.3. Добавление в раскадровку контроллера с панелью вкладок

## Постановка задачи

С помощью раскадровок требуется создать приложение, построенное на базе контроллера с панелью вкладок.


**Рис. 6.8.** Наш текст успешно отобразился в качестве заголовка второго контроллера вида

## Решение

Создайте в Xcode приложение с единственным видом и встройте первый контроллер вида в контроллер вида с панелью вкладок. Чтобы сделать это, выполните следующие шаги.

1. В конструкторе интерфейса выберите в раскадровке контроллер вида. В меню Editor (Редактор) выберите пункт Embed in (Встроить), а затем выберите Tab Bar Controller (Контроллер с панелью вкладок) (рис. 6.9).
2. Перейдите в библиотеку объектов контроллера интерфейсов, найдите там новый экземпляр контроллера вида и перетащите его в раскадровку.

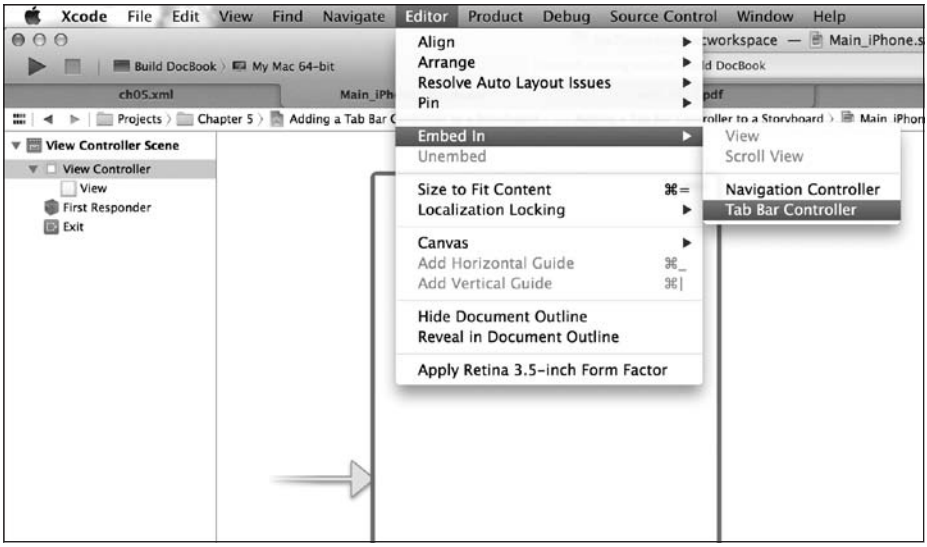


**Рис. 6.9.** Встраивание корневого контроллера вида в контроллер с панелью вкладок

3. Удерживая нажатой клавишу Ctrl, перетащите указатель мыши из контроллера с вкладками в созданный вами контроллер вида (рис. 6.10). Откроется диалоговое окно, в котором следует найти раздел Relationship Segues (Взаимосвязи через сегвеи) и выбрать в нем View Controller (Контроллер вида) (рис. 6.11).

Теперь запустите приложение в эмуляторе iOS. В нижней части экрана вы увидите два элемента (рис. 6.12). Каждый из этих элементов соответствует одному из ваших контроллеров вида. Если бы это контроллерное приложение с вкладками создавал я, то я поместил бы в каждой из вкладок по навигационному контроллеру. Если вы также хотите задействовать такую возможность, воспользуйтесь приемами, изученными в разделе 6.1, и встройте контроллеры видов в навигационные контроллеры (рис. 6.13).

## См. также

Раздел 6.1.

# 6.4. Внедрение специальных переходов между сегвеями в раскадровке

## Постановка задачи

Требуется внедрить и использовать в файлах раскадровки новый тип перехода, чтобы перемещение из одного контроллера вида в другой выполнялось специально заданным образом — например, с применением пользовательской анимации.

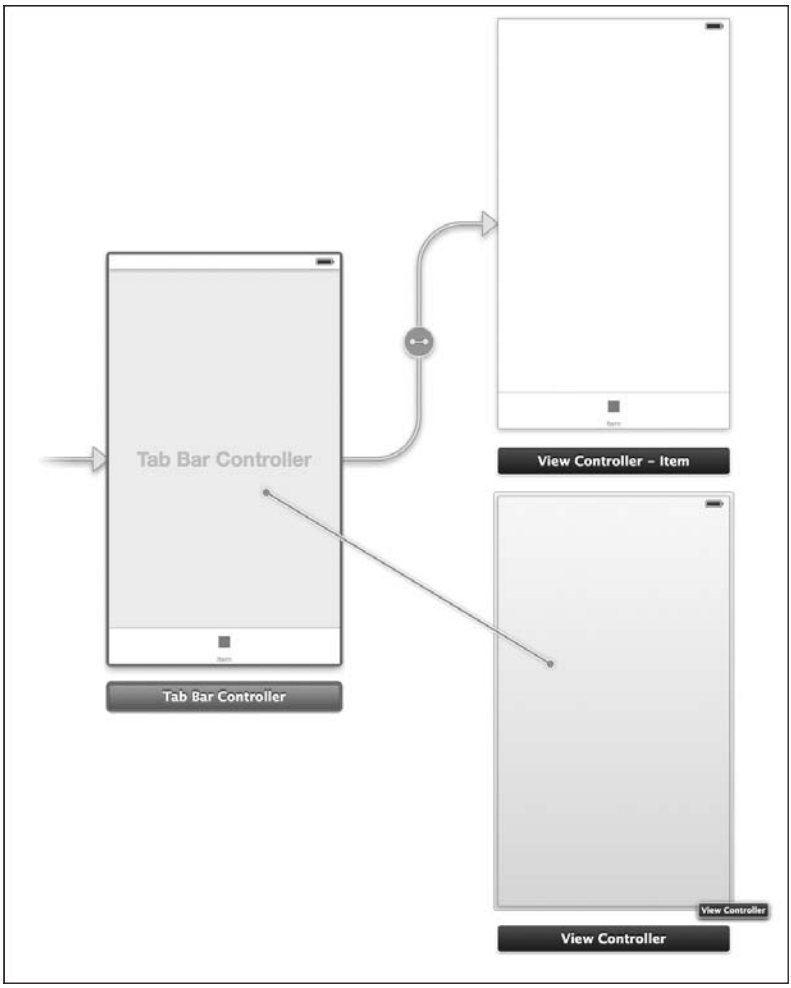


**Рис. 6.10.** Соединение контроллера вида с контроллером, имеющим панель вкладок

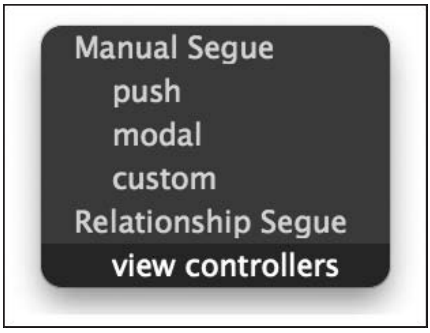


**Рис. 6.11.** Ассоциирование контроллера вида с массивом контроллеров видов, находящихся в контроллере с панелью вкладок


**Рис. 6.12.** Контроллеры вида успешно отображаются в контроллере с панелью вкладок

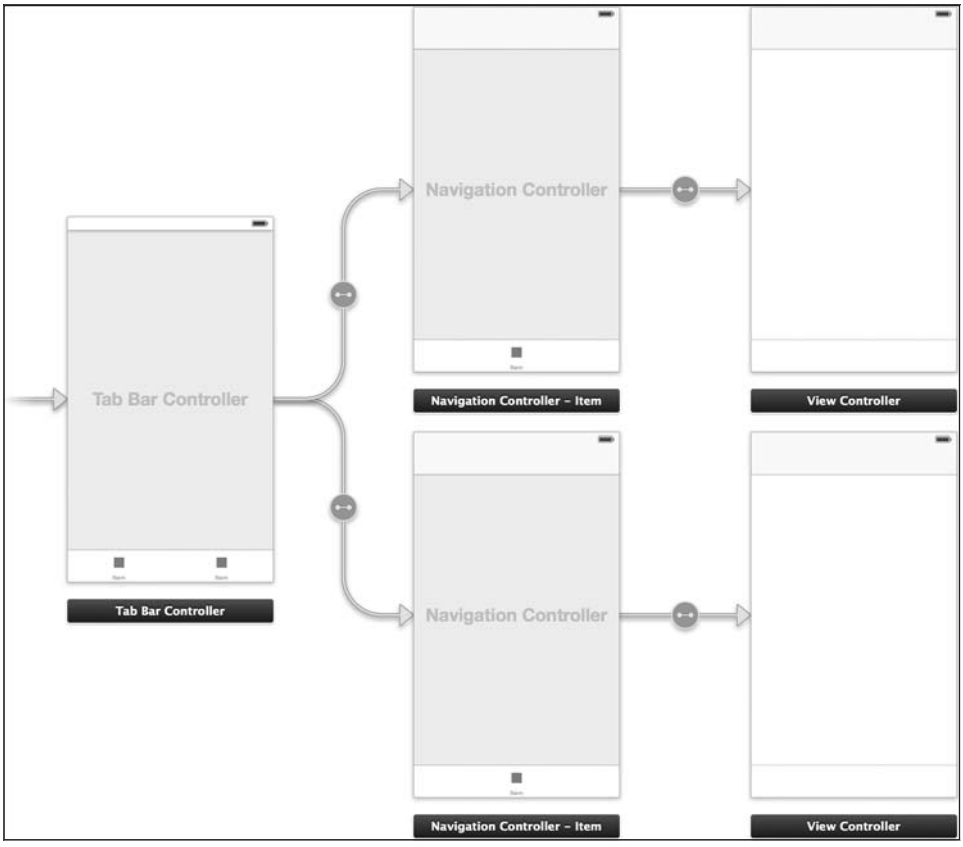


**Рис. 6.13.** Встраивание ваших контроллеров видов в навигационные контроллеры, расположенные в контроллере панели вкладок

## Решение

Создайте подкласс от `UIStoryboardSegue` и переопределите его метод `perform` в соответствии со стоящими перед вами задачами.

## Обсуждение

По умолчанию в раскадровках предлагается несколько полезных типов сегвеев, например пуш-сегвеи и модальные сегвеи. Они, конечно, очень удобны, но иногда

требуется выполнять переход от одного вида к другому каким-то особым образом. В таких случаях лучше всего воспользоваться специальным сегвеем. Итак, создадим сегвей. В первом контроллере вида мы разрешаем переход во второй контроллер вида, причем на экране этот переход будет выглядеть как перелистывание. Для решения этой задачи выполните следующие шаги.

1. Создайте в Xcode проект, основанный на шаблоне Single View Application (Приложение с единственным видом).
2. В файле раскадровки создайте второй контроллер вида и поместите кнопку в центр первого контроллера вида. Удерживая нажатой клавишу Ctrl, перетащите указатель с экранной кнопки на второй контроллер вида. На этом этапе на экране откроется диалоговое окно, где система запросит информацию о типе перехода, который вы хотите ассоциировать с данным сегвеем. В этом диалоговом окне выберите вариант Custom (Специальный) (рис. 6.14).


**Рис. 6.14.** Ассоциирование специального сегвея с действием, выполняемым кнопкой

3. Теперь выберите желаемый сегвей и в инспекторе атрибутов (Attribute Inspector) конструктора интерфейсов измените имя класса сегвея на MySegue (рис. 6.15). Этого класса пока не существует, но не волнуйтесь — мы напишем его уже в данном разделе.
4. Теперь создайте в Xcode новый класс Objective-C внутри вашего проекта, назовите этот класс MySegue (именно такое имя вы присвоили ему на предыдущем этапе) и убедитесь в том, что этот класс наследует от `UIStoryboardSegue`. Как только система создаст для вас этот класс, реализуйте его метод `perform` следующим образом:

```
#import "MySegue.h"

@implementation MySegue

- (void) perform{

  UIViewController *source = self.sourceViewController;
  UIViewController *destination = self.destinationViewController;

[UIView transitionFromView:source.view
```

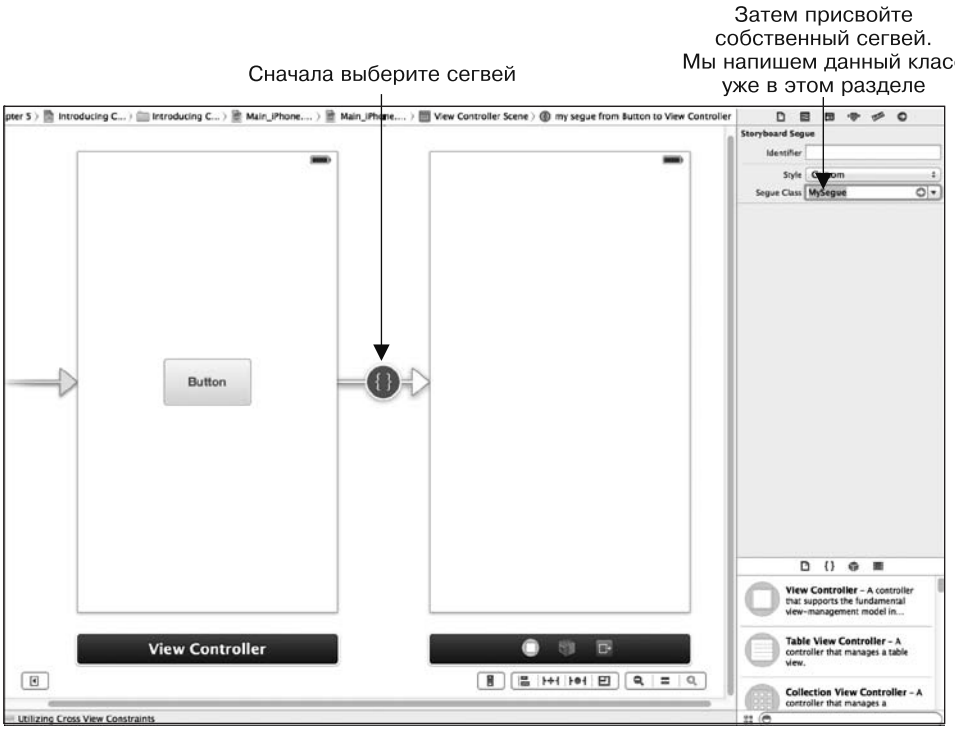


**Рис. 6.15.** Присваивание сегвею нашего собственного имени класса

```
                toView:destination.view
              duration:0.50f
               options:UIViewAnimationOptionTransitionFlipFromTop
            completion:^(BOOL finished) {
              NSLog(@"Transitioning is finished");
            }];
}
@end
```

Вот и все. Теперь можете запустить приложение и убедиться в том, что при нажатии кнопки в первом контроллере вида запустится специальный сегвей, который, в свою очередь, отобразит на экране второй контроллер вида. В этом примере мы используем для реализации перехода метод класса `transitionFromView:toView:duration:options:completion:`, относящийся к классу `UIView`. Этот метод принимает довольно много параметров, которые описаны далее:

- `transitionFromView` — вид, с которого должен начинаться переход. В контексте нашего сегвея это вид с исходным контроллером вида;
- `toView` — это целевой вид, к которому должен привести переход. В нашем сегвее это вид с целевым контроллером вида;
- `duration` — длительность анимации в секундах;

- `options` — тип анимации, которую вы хотите выполнить. Это значение типа `UIViewAnimationOptions`. Если вы хотите просмотреть все параметры, доступные здесь, нажмите на клавиатуре Command+Shift+O, введите `UIViewAnimationOptions`, а потом нажмите клавишу Enter;
- `completion` — блок завершения, вызываемый сразу же по окончании перехода.

Прежде чем завершить этот раздел, необходимо упомянуть еще об одной важной вещи. Работа, которую вы выполняете (специальный переход), должна осуществляться в методе экземпляра `perform` специального класса сегвея. Это, в частности, означает, что вы не можете вывести в этом методе окно с предупреждением и отобразить его для пользователя (а также рассчитывать на то, что пользователь сможет нажать кнопку Yes или No в зависимости от своего решения *и только потом* выполнить переход). Это не сработает. Поэтому сначала подумайте, чего вы хотите добиться в вашем сегвее и является ли создание подкласса от `UIStoryboardSegue` наилучшим выходом из ситуации.

## См. также

Разделы 6.0 и 6.1.

# 6.5. Размещение изображений и других компонентов пользовательского интерфейса в раскадровках

## Постановка задачи

Требуется размещать в файлах раскадровок изображения, кнопки и другие компоненты пользовательского интерфейса.

## Решение

Воспользуйтесь библиотекой объектов из конструктора интерфейса и найдите в ней разные компоненты для пользовательского интерфейса. Когда будете готовы поместить их в файлы раскадровок, просто перетащите туда эти элементы. Затем вы сможете сконфигурировать эти компоненты в инспекторе атрибутов.

## Обсуждение

Допустим, вы хотите разместить в раскадровке несколько изображений. Находясь в конструкторе интерфейса, где уже должен быть открыт файл раскадровки, нажмите комбинацию клавиш Ctrl+Alt+Command+3 и таким образом перейдите в библиотеку объектов. Здесь найдите компонент Image View (Вид с изображением) (рис. 6.16) и перетащите его в основной вид с контроллером. Теперь одновременно нажмите на клавиатуре Alt+Command+4, чтобы открыть инспектор атрибутов.

На этой панели вы сможете сконфигурировать вид с изображением. Чтобы поместить изображение в этот вид, просто добавьте изображение к вашему проекту. Пока вид с изображением остается выделенным, в инспекторе атрибутов задайте для этого вида свойство image (рис. 6.17).

Периодически могут возникать ситуации, когда вы просто не можете найти в библиотеке объектов нужный компонент, но уверены, что он существует. Со мной это тоже случалось. В библиотеке объектов есть очень удобная строка для поиска, в которой вы можете написать имя интересующего вас компонента. Чтобы попасть в строку поиска, убедитесь, что вы уже открыли библиотеку объектов (для этого требуется нажать сочетание клавиш Ctrl+Alt+Command+3), а затем Command+Alt+L (рис. 6.18).

В инспекторе атрибутов можно сконфигурировать большинство важнейших свойств различных компонентов пользовательского интерфейса, которые вы помещаете в раскадровки. Тем не менее для решения определенных задач не обойтись без написания кода.

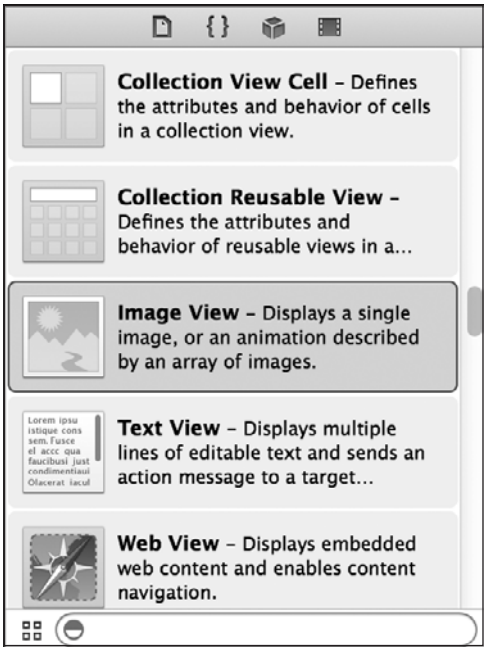


**Рис. 6.16.** Вид с изображением, служащий компонентом пользовательского интерфейса (так он выглядит в библиотеке объектов)

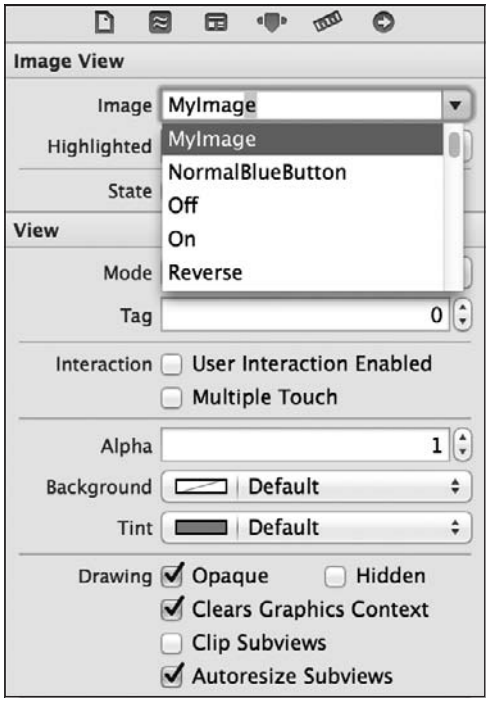


**Рис. 6.17.** Установка свойства-изображения для вида с изображением в инспекторе атрибутов в конструкторе интерфейса

## См. также

Раздел 6.0.


**Рис. 6.18.** Строка поиска в библиотеке объектов позволяет быстро найти интересующий объект

# 7 Параллелизм

## 7.0. Введение

*Параллелизм* (конкурентное исполнение программ) возникает, когда одновременно выполняется несколько задач. Современные операционные системы позволяют параллельно выполнять задачи даже на одном процессоре. Такая возможность гарантируется, если выделить на каждую задачу строго определенный промежуток (квант) процессорного времени. Например, если за секунду требуется решить 10 задач и все они имеют одинаковый приоритет, то операционная система разделит 1000 мс на 10 и уделит решению каждой задачи 100 мс процессорного времени. Таким образом, все эти задачи решаются в течение одной секунды, и пользователю кажется, что это происходит параллельно.

При этом технологии развиваются, и теперь в нашем распоряжении есть процессоры с несколькими ядрами. Это означает, что один процессор действительно может решать несколько задач одновременно. Операционная система диспетчеризует задачи к процессору и дожидается, пока они будут выполнены. Вот так все просто!

Grand Central Dispatch (GCD) — это низкоуровневый API, написанный на языке C и работающий с блоковыми объектами. GCD отлично приспособлен для направления различных задач нескольким ядрам, так что программист может не задумываться о том, какое ядро решает какую задачу. Многоядерные устройства с операционной системой Mac OS X, в частности ноутбуки, имеются в свободном доступе уже довольно давно. А с появлением таких многоядерных устройств, как новый iPad, мы можем писать и интересные многопоточные приложения для системы iOS, рассчитанные на работу с несколькими ядрами.

Центральной составляющей GCD являются диспетчерские очереди. Диспетчерские очереди, как мы вскоре увидим, представляют собой пулы потоков, управляемые GCD в базовой операционной системе, будь то iOS или Mac OS X. Вы не будете работать с этими потоками напрямую. Вы будете иметь дело только с диспетчерскими очередями, распределяя *задачи* по этим очередям и приказывая очередям инициировать решение задач. GCD предлагает несколько режимов решения задач: синхронно, асинхронно, с определенной задержкой и т. д.

Чтобы приступить к использованию GCD в ваших приложениях, в проект не требуется импортировать каких-либо специальных библиотек. Apple уже встроила GCD в различные фреймворки, в частности в Core Foundation и Cocoa/Cocoa Touch. Все методы и типы данных, имеющиеся в GCD, начинаются с ключевого слова `dispatch_`. Например, `dispatch_async` позволяет направить задачу в очередь для асинхронного выполнения, а `dispatch_after` — выполнить блок кода после определенной задержки.

До того как появился GCD, программисту приходилось создавать собственные потоки для параллельного решения задач. Примерно такой поток разработчик iOS создаст для того, чтобы выполнить определенную операцию 1000 раз:

```
- (void) doCalculation{
  /* Здесь происходят вычисления. */
}

- (void) calculationThreadEntry{

  @autoreleasepool {
    NSUInteger counter = 0;
    while ([[NSThread currentThread] isCancelled] == NO){
      [self doCalculation];
      counter++;
      if (counter >= 1000){
        break;
      }
    }
  }

}

- (BOOL)            application:(UIApplication *)application
  didFinishLaunchingWithOptions:(NSDictionary *)launchOptions{

  /* Начинаем поток. */
  [NSThread detachNewThreadSelector:@selector(calculationThreadEntry)
                           toTarget:self
                         withObject:nil];

  self.window = [[UIWindow alloc] initWithFrame:
                 [[UIScreen mainScreen] bounds]];
  self.window.backgroundColor = [UIColor whiteColor];
  [self.window makeKeyAndVisible];
  return YES;
}
```

Программист должен создать поток вручную, а потом придать ему требуемую структуру (точку входа, автоматически высвобождаемый пул и основной цикл потока). Когда мы пишем аналогичный код с помощью GCD, нам на самом деле приходится сделать не так уж много. Мы просто помещаем наш код в блоковый объект и направляем этот блок в GCD для выполнения. Где именно будет выпол-

няться данный код — в главном потоке или в каком-нибудь другом, — зависит именно от нас. Вот пример:

```
dispatch_queue_t queue =
  dispatch_get_global_queue(DISPATCH_QUEUE_PRIORITY_DEFAULT, 0);

size_t numberOfIterations = 1000;
  dispatch_async(queue, ^(void) {
    dispatch_apply(numberOfIterations, queue, ^(size_t iteration){
      /* Здесь выполняется операция. */
    });
});
```

В этой главе будет рассказано обо всем, что нужно знать о GCD. Здесь вы научитесь писать современные многопоточные приложения для iOS и Mac OS X, помогающие достичь впечатляющей производительности на таких многоядерных устройствах, как iPad 2.

Мы довольно много будем работать с диспетчерскими очередями, поэтому необходимо досконально разобраться с теми концепциями, которые лежат в их основе. Диспетчерские очереди бывают трех типов.

- *Главная очередь* — занимается выполнением всех задач из главного потока, и именно здесь Cocoa и Cocoa Touch требуют от программиста вызывать все методы, относящиеся к пользовательскому интерфейсу. Пользуйтесь функцией `dispatch_get_main_queue`, помогающей управлять главной очередью.
- *Параллельные очереди* — это очереди, которые можно получать из GCD для выполнения синхронных или асинхронных задач. В нескольких параллельных очередях могут одновременно выполняться несколько задач, причем с завидной легкостью. Представляете, больше никакого управления потоками, ур-р-ра! Пользуйтесь функцией `dispatch_get_global_queue`, помогающей управлять параллельными очередями.
- *Последовательные очереди* — всегда выполняют поставленные в них задачи по принципу «первым пришел — первым обслужен» (First In First Out, FIFO). При этом не имеет значения, являются эти задачи синхронными или асинхронными. Такой принцип работы означает, что последовательная очередь может выполнять в любой момент только один блок кода. Однако такие очереди *не* применяются в главном потоке, поэтому отлично подходят для решения задач, которые должны выполняться в строгом порядке и не блокировать при этом главный поток. Чтобы создать последовательную очередь, пользуйтесь функцией `dispatch_queue_create`.

Существуют два механизма отправки задач в диспетчерские очереди:

- блочные объекты (см. раздел 7.1);
- функции C.

Блочные объекты позволяют наиболее эффективно использовать GCD и его огромный потенциал. Некоторые функции GCD были расширены таким образом, чтобы программист мог использовать функции C вместо блочных объектов. Однако в действительности лишь небольшое подмножество GCD-функций допускают

замену объектов функциями C, поэтому перед дальнейшим изучением материала обязательно ознакомьтесь с разделом о блочных объектах (разделом 7.1).

Функции C, предоставляемые различным GCD-функциям, должны относиться к типу `dispatch_function_t`. Вот как этот тип определяется в библиотеках Apple:

```
typedef void (*dispatch_function_t)(void *);
```

Итак, если мы хотим, например, создать функцию под названием `myGCDFunction`, потребуется реализовать ее следующим образом:

```
void myGCDFunction(void * paramContext){

  /* Вся работа выполняется здесь */

}
```

Параметр paramContext относится к контексту, который GCD позволяет передавать C-функциям при диспетчеризации задач к этим функциям. Вскоре мы подробно об этом поговорим.

Блочные объекты, передаваемые GCD-функциям, не всегда имеют одинаковую структуру. Некоторые должны принимать параметры, другие — нет, но ни один блочный объект, передаваемый GCD, не возвращает значения.

В любой момент в ходе жизненного цикла приложения вы можете одновременно задействовать несколько диспетчерских очередей. В системе есть только одна основная очередь, но вы сами можете создать сколько угодно последовательных диспетчерских очередей (конечно, в разумных пределах) для любых функций, которые, возможно, понадобится реализовать в вашем приложении. Кроме того, можно получить несколько параллельных очередей и направить им ваши задачи. Задачи можно передавать диспетчерским очередям двумя способами: как блоковые объекты и как функции языка C, о чем рассказано ранее.

Блоковые объекты — это *пакеты* с кодом, которые в Objective-C обычно имеют форму методов. Блоковые объекты вместе с GCD образуют гармоничную среду, в которой можно создавать высокопроизводительные многопоточные приложения для iOS и Mac OS X. Вы можете спросить: «А что же такого особенного в блоковых объектах и GCD?» Ответ прост: больше никаких потоков! Все, что от вас требуется, — поместить код в блоковые объекты и перепоручить GCD выполнение этого кода.

Вероятно, важнейшая разница между блоковыми объектами и традиционными указателями на функции заключается в том, что блоковые объекты копируют значения локальных переменных, доступ к которым происходит внутри блокового объекта, и сохраняют эти копии для локального использования. Если значения этих переменных изменяются вне области видимости блокового объекта, вы тем не менее можете быть уверены в том, что в блоковом объекте сохранилась собственная копия переменной. Вскоре мы обсудим эти вопросы подробнее.

Блоковые объекты в Objective-C — это сущности, которые в среде программистов принято называть *объектами первого класса*. Это означает, что вы можете создавать код динамически, передавать блоковый объект методу в качестве параметра и воз-

вращать блоковый объект от метода. Все это позволяет более уверенно выбирать, что вы хотите делать во время исполнения и изменять ход действия программы. В частности, GCD может выполнять блоковые объекты в отдельных потоках. Поскольку блоковые объекты являются объектами Objective-C, с ними можно обращаться как с любыми другими объектами.

Иногда блоковые объекты называются замкнутыми выражениями (Closures).

Создание блоковых объектов напоминает создание обычных функций языка C, как будет показано в разделе 7.1. Блоковые объекты могут возвращать значения и принимать параметры. Блоковые объекты можно определять как встраиваемые либо обрабатывать как отдельные блоки кода наподобие функций языка C. При создании встраиваемым способом область видимости переменных, доступных блоковым объектам, существенно отличается от аналогичной области видимости, если блоковый объект реализуется как отдельный блок кода.

GCD работает с блоковыми объектами. При выполнении задач с помощью GCD вы можете передавать блоковый объект, код которого будет выполняться синхронно или асинхронно в зависимости от того, какие методы используются в GCD. Следовательно, вы можете создать блоковый объект, который будет отвечать за загрузку данных по URL (универсальному идентификатору ресурса), передаваемому в качестве параметра. Такой отдельный блоковый объект можно синхронно или асинхронно использовать в различных местах приложения на ваш выбор. Не требуется делать блоковый объект синхронным или асинхронным по сути. Вы всего лишь будете вызывать его с помощью тех или иных методов, относящихся к GCD, — синхронных или асинхронных, — и блоковый объект будет *просто работать*.

Блоковые объекты — довольно новое явление для программистов, создающих приложения для iOS и Mac OS X. На самом деле блоковые объекты пока еще уступают по популярности потокам, поскольку их синтаксис несколько отличается от организации обычных методов Objective-C и более сложен. Тем не менее потенциал блоковых объектов огромен, и Apple довольно активно внедряет их в свои библиотеки. Такие дополнения уже можно заметить в некоторых классах, например `NSMutableArray`. Здесь программист может сортировать массив с помощью блокового объекта.

Эта глава целиком посвящена созданию и использованию блоковых объектов в приложениях для iOS и Mac OS X, использованию GCD для передачи задач операционной системе, а также работе с потоками и таймерами. Я хотел бы подчеркнуть, что единственный способ освоить синтаксис блоковых объектов — написать несколько таких объектов самостоятельно. Изучите код примеров, которые сопровождают эту главу, и попробуйте реализовать собственные блоковые объекты.

В данной главе будут рассмотрены базовые вопросы, связанные с блоковыми объектами, а потом мы обсудим некоторые более сложные темы. В частности, поговорим об интерфейсе Grand Central Dispatch, о потоках, таймерах, операциях и очередях операций. Вы усвоите все, что необходимо знать о блоковых объектах, а потом перейдете к материалу о GCD. По моему опыту, лучше всего изучать

блоковые объекты на примерах, поэтому данная глава изобилует примерами. Обязательно опробуйте их в Xcode, чтобы по-настоящему *усвоить* синтаксис блоковых объектов.

Операции конфигурируются для синхронного или асинхронного запуска блока кода. Можно управлять операциями вручную либо помещать их в *очереди операций*, которые обеспечивают параллельное исполнение, избавляя вас от необходимости заниматься управлением потоками в фоновом режиме. В этой главе мы рассмотрим, как работать с операциями или очередями операций, а также с простыми потоками и таймерами, чтобы синхронно и асинхронно решать задачи и запускать приложения.

В Сосоа выполняются операции трех типов:

- *блоковые операции* — обеспечивают выполнение одного или нескольких блоковых объектов;
- *активизирующие операции* — позволяют активизировать метод в другом, уже существующем объекте;
- *обычные операции* — это классы обычных операций, от которых необходимо создавать подклассы. Код, который нужно выполнить, следует писать в методе `main` объекта операции.

Как уже упоминалось ранее, управлять операциями можно с помощью очередей операций, относящихся к типу данных `NSOperationQueue`. После инстанцирования операции любого из упомянутых типов (блоковой, активизирующей или обычной) их можно добавить в очередь операций и перепоручить управление операцией самой очереди.

Объект операции может зависеть от других объектов операций. Можно приказать объекту операции дождаться, пока не выполнится одна или несколько иных операций, и только после этого приказать ему решить ассоциированную с ним задачу. Без применения зависимостей вы никак не сможете влиять на порядок выполнения операций. Так, если добавить операции в очередь в определенном порядке, это не гарантирует выполнения операций в том же порядке, несмотря на то что термин *«очередь»* это как бы подразумевает.

Работая с операциями и очередями операций, не следует забывать о нескольких важных вещах.

- По умолчанию операция выполняется в том потоке, который ее начал с помощью их метода экземпляра `start`. Если вы хотите, чтобы операции выполнялись асинхронно, нужно использовать либо очередь операций, либо подкласс `NSOperation` и выделить новый поток в методе экземпляра `main`, относящегося к операции.
- Операция может дождаться окончания выполнения другой операции и только после этого начаться. Будьте осторожны и не создавайте взаимозависимых операций. Такая распространенная ошибка называется *взаимоблокировкой*, или *клинчем* (Deadlock). Иными словами, нельзя ставить операцию A в зависимость от операции B, если операция B уже зависит от операции A. В таком случае они обе будут ждать вечно, расходуя память и, возможно, вызывая зависание приложения.

- Операции можно отменять. Так, если вы создаете подклассы от `NSOperation`, чтобы делать собственные виды объектов операций, обязательно пользуйтесь методом экземпляра `isCancelled`. Он применяется, чтобы проверить, не была ли отменена определенная операция, прежде чем переходить к выполнению задачи, связанной с этой операцией. Например, если задача вашей операции — проверять доступность соединения с Интернетом раз в 20 с, то перед каждым запуском операции нужно вызвать метод экземпляра `isCancelled`, чтобы сначала убедиться, что операция не отменена, и только после этого пытаться проверять наличие соединения с Интернетом. Если на выполнение операции уходит более нескольких секунд (например, если это загрузка файла), то при выполнении задачи нужно также периодически проверять метод `isCancelled`.
- Объекты операций обязаны выполнять «уведомление наблюдателей об изменениях в свойствах наблюдаемого объекта» (KVO, Key-Value Observing) на различных ключевых путях, в частности `isFinished`, `isReady` и `isExecuting`. В одной из следующих глав мы обсудим механизм KVO, а также KVC — механизм для доступа к полям объекта по именам этих полей.
- Если вы планируете создавать подкласс от `NSOperation` и выполнять специальную реализацию для операции, вам следует создать собственный автоматически высвобождаемый пул в методе `main`, относящемся к операции. Данный метод вызывается из метода `start`. Эти вопросы мы подробнее рассмотрим далее в этой главе.
- Всегда сохраняйте ссылки на создаваемые вами объекты операций. Сама параллельная природа, присущая очередям операций, исключает возможность получения ссылки на операцию после того, как она добавлена в очередь.

Потоки и таймеры — это объекты, являющиеся подклассами от `NSObject`. Для порождения потока выполняется больше работы, чем для создания таймеров, а настройка цикла потока — более сложная задача, чем обычное слушание таймера, запускающего селектор. Когда приложение работает в операционной системе iOS, система создает для этого приложения как минимум один поток. Этот поток называется главным (Main Thread). Все потоки и таймеры должны добавляться в цикл исполнения (Run Loop). Цикл исполнения, как понятно из его названия, — это цикл, в ходе которого могут происходить разные события, например запуск таймера или выполнение потока. Обсуждение циклов исполнения выходит за рамки этой главы, но иногда я буду упоминать такой цикл.

Цикл исполнения — это, в сущности, обычный цикл, у которого есть начальная точка, условие завершения и серия событий, которые необходимо обработать в ходе этого цикла. Поток или таймер прикрепляются к циклу исполнения, и, в сущности, именно они заставляют цикл исполнения работать.

Главный поток приложения — это тот поток, который обрабатывает события пользовательского интерфейса. Если вы выполняете в главном потоке долговременную задачу, то быстро станет заметно, что интерфейс перестает отвечать на запросы или реагирует медленно. Во избежание этого можно создавать отдельные потоки и/или таймеры, каждый из которых выполняет собственную задачу (даже если она сравнительно долговременная). Но при этом главный поток не будет блокироваться.

# 7.1. Создание блоковых объектов

## Постановка задачи

Необходимо иметь возможность писать собственные блоковые объекты либо использовать блоковые объекты с классами из iOS SDK.

## Решение

Просто необходимо понимать базовую разницу между синтаксисом блоковых объектов и синтаксисом классических функций языка C. Эта разница рассматривается в подразделе «Обсуждение» данного раздела.

## Обсуждение

Блоковые объекты могут быть либо встраиваемыми, либо записываемыми как отдельные блоки кода. Начнем с объектов второго типа. Предположим, у нас есть метод языка Objective-C, принимающий два целочисленных значения типа `NSInteger` и возвращающий разницу двух этих значений в форме `NSInteger`. Разница получается в результате вычитания одного значения из другого:

```
- (NSInteger) subtract:(NSInteger)paramValue
                  from:(NSInteger)paramFrom{

  return paramFrom - paramValue;

}
```

Очень просто, правда? Теперь преобразуем этот код Objective-C в классическую функцию языка C, обеспечивающую такую же функциональность. Это еще на шаг приблизит нас к пониманию синтаксиса блоковых объектов:

```
NSInteger subtract(NSInteger paramValue, NSInteger paramFrom){

  return paramFrom - paramValue;

}
```

Как видите, синтаксис функции на C значительно отличается от синтаксиса аналогичной функции на языке Objective-C. Теперь рассмотрим, как можно написать ту же функцию в виде блокового объекта:

```
NSInteger (^subtract)(NSInteger, NSInteger) =
  ^(NSInteger paramValue, NSInteger paramFrom){

    return paramFrom - paramValue;

};
```

Прежде чем перейти к детальному описанию синтаксиса блоковых объектов, приведу еще несколько примеров. Предположим, что у нас есть функция на языке C, принимающая параметр типа `NSUInteger` (беззнаковое целое число) и возвращающая строку типа `NSString`. Вот как данная функция реализуется на C:

```
NSString* intToString (NSUInteger paramInteger){

  return [NSString stringWithFormat:@"%lu",
          (unsigned long)paramInteger];

}
```

Чтобы научиться форматировать строки с применением системонезависимых указателей формата на языке Objective-C, ознакомьтесь с String Programming Guide in the iOS Developer Library (Руководство по программированию строк в библиотеке разработчика iOS). Адрес документа на сайте Apple: https://developer.apple.com/library/ios/#documentation/Cocoa/Conceptual/Strings/Articles/formatSpecifiers.html.

Блоковый объект, эквивалентный данной функции языка C, показан в примере 7.1.

**Пример 7.1.** Образец блокового объекта, определенного в виде функции

```
NSString* (^intToString)(NSUInteger) = ^(NSUInteger paramInteger){
  NSString *result = [NSString stringWithFormat:@"%lu",
                      (unsigned long)paramInteger];

  return result;
};
```

Простейший независимый блоковый объект — это блоковый объект, возвращающий `void` и не принимающий никаких параметров:

```
void (^simpleBlock)(void) = ^{
  /* Здесь реализуется блоковый объект. */
};
```

Блоковые объекты инициируются точно так же, как и функции на языке C. Если у них есть какие-либо параметры, то вы передаете их так, как и в функции C. Любое возвращаемое значение можно получить точно так же, как и возвращаемое значение функции на языке C. Вот пример:

```
NSString* (^intToString)(NSUInteger) = ^(NSUInteger paramInteger){
  NSString *result = [NSString stringWithFormat:@"%lu",
                      (unsigned long)paramInteger];
  return result;
};

- (void) callIntToString{

  NSString *string = intToString(10);
  NSLog(@"string = %@", string);

}
```

Метод `callIntToString` языка Objective-C вызывает блоковый объект `intToString`, передавая этому блоковому объекту в качестве единственного параметра значение `10` и помещая возвращаемое значение данного блокового объекта в локальную переменную `string`.

Теперь, когда мы знаем, как писать блоковые объекты в виде независимых блоков кода, рассмотрим передачу блоковых объектов как передачу параметров методам языка Objective-C. Чтобы понять смысл следующего примера, нужно прибегнуть к определенным абстракциям.

Предположим, у нас есть метод Objective-C, принимающий целое число и выполняющий над ним какое-либо преобразование. Такое преобразование может меняться в зависимости от того, что еще происходит в программе. Мы уже знаем, что у нас будет целое число в качестве ввода и строка в качестве вывода, но сам процесс преобразования поручим блоковому объекту — а этот объект может быть иным при каждом вызове метода. Следовательно, в качестве параметров данный метод будет принимать и целое число, которое необходимо преобразовать, и тот блок, который будет выполнять преобразование.

Для блокового объекта воспользуемся тем же блоковым объектом `intToString`, который мы реализовали в примере 7.1. Теперь нам нужен метод на языке Objective-C, который будет принимать в качестве параметра беззнаковое целое число, а в качестве еще одного параметра — блоковый объект. С беззнаковым целым в качестве параметра все просто, но как сообщить методу, что он должен принимать блоковый объект *того же типа*, к которому относится блоковый объект `intToString`? Сначала определяем псевдоним сигнатуры блокового объекта `intToString` (с помощью ключевого слова `typedef`) и таким образом сообщаем компилятору, какие параметры должен принимать блоковый объект:

```
typedef NSString* (^IntToStringConverter)(NSUInteger paramInteger);
```

Объявление `typedef` просто сообщает компилятору, что блоковые объекты, принимающие в качестве параметра целое число и возвращающие строку, можно представлять с помощью обычного идентификатора, называемого `IntToStringConverter`. Итак, пойдем дальше и напишем метод на Objective-C, который будет принимать в качестве параметров и целое число, и блоковый объект типа `IntToStringConverter`:

```
- (NSString *) convertIntToString⊗NSUInteger)paramInteger
                usingBlockObject⊗IntToStringConverter)paramBlockObject{

  return paramBlockObject(paramInteger);

}
```

Теперь требуется просто вызвать метод `convertIntToString:`, сопровождаемый объектом на наш выбор (пример 7.2).

**Пример 7.2.** Вызов блокового объекта в другом методе

```
- (void) doTheConversion{

  NSString *result = [self convertIntToString:123
```

```
                                usingBlockObject:intToString];

  NSLog(@"result = %@", result);

}
```

Теперь, когда мы немного разбираемся в независимых блоковых объектах, поговорим о встраиваемых блоковых объектах. В только что рассмотренном методе `doTheConversion` мы передавали методу `convertIntToString:usingBlockObject:` в качестве параметра блоковый объект `intToString`. Что если бы у нас не было в распоряжении готового блокового объекта, который можно было бы передать этому методу? На самом деле это не доставило бы нам никаких проблем. Как уже упоминалось, блоковые объекты — это функции первого класса и их можно создавать во время исполнения. Рассмотрим альтернативную реализацию метода `doTheConversion` (пример 7.3).

**Пример 7.3.** Блоковый объект, определенный в виде функции

```
- (void) doTheConversion{

  IntToStringConverter inlineConverter = ^(NSUInteger paramInteger){
    NSString *result = [NSString stringWithFormat:@"%lu",
                        (unsigned long)paramInteger];
    return result;
  };

  NSString *result = [self convertIntToString:123
                              usingBlockObject:inlineConverter];

  NSLog(@"result = %@", result);

}
```

Сравните примеры 7.1 и 7.3. Я удалил имевшийся в первом варианте код, в котором мы формировали сигнатуру блокового объекта. Данная сигнатура состояла из имени и аргумента — `(^intToString) (NSUInteger)`. Остальную часть блокового объекта я не трогаю, и теперь он становится анонимным объектом. Но это не означает, что я никак не могу сослаться на блоковый объект. С помощью знака равенства (=) я присваиваю блоковый объект типу и имени: `IntToStringConverter inlineConverter`. Теперь я могу воспользоваться типом данных, чтобы стимулировать правильную работу методов, а при самой операции передачи блокового объекта использовать его имя.

Кроме того способа создания встраиваемых блоковых объектов, который только что был продемонстрирован, существует способ создания блокового объекта *на этапе передачи* его как параметра:

```
- (void) doTheConversion{

  NSString *result =
  [self convertIntToString:123
          usingBlockObject:^NSString *(NSUInteger paramInteger) {

    NSString *result = [NSString stringWithFormat:@"%lu",
```

```
                        (unsigned long)paramInteger];
    return result;

  }];

  NSLog(@"result = %@", result);

}
```

Сравните этот пример с примером 7.2. Оба метода используют блоковый объект с применением синтаксиса `usingBlockObject`. Но, в то время как при применении первого варианта мы ссылались по имени на предварительно определенный блоковый объект (`intToString`), во втором варианте блоковый объект создается на лету. В этом коде мы создали встраиваемый блоковый объект, который передается методу `convertIntToString:usingBlockObject:` как второй параметр.

# 7.2. Доступ к переменным в блоковых объектах

## Постановка задачи

Необходимо понять разницу между доступом к переменным в методах Objective-C и доступом к этим переменным в блоковых объектах.

## Решение

Вот краткое обобщение того, что необходимо знать о переменных в блоковых объектах.

- Локальные переменные в блоковых объектах работают точно так же, как и в методах Objective-C.
- При работе со встраиваемыми блоковыми объектами к локальным относятся не только те переменные, которые определены внутри блока, но и те, что определены в методе, реализующем данный блоковый объект (чуть позже рассмотрим примеры).
- Нельзя ссылаться на `self` в независимых блоковых объектах, реализованных в классе Objective-C. Если необходим доступ к `self`, то вам нужно передать его объект блоковому объекту в качестве параметра. Чуть позже рассмотрим на примере и такую ситуацию.
- Во встраиваемом блоковом объекте на `self` можно ссылаться лишь в тех случаях, когда `self` присутствует в лексической области видимости, в рамках которой и создается блоковый объект.
- При работе со встраиваемыми блоковыми объектами локальные переменные, определяемые *внутри* реализации блокового объекта, доступны для считывания, но не для записи. Однако есть и исключение. Блоковый объект может записывать

информацию в такие переменные, если они определены с типом хранения __block. Пример мы также рассмотрим.

- Предположим, у вас есть блоковый объект типа NSObject, а внутри реализации этого объекта вы используете блоковый объект с GCD. Внутри данного блокового объекта у вас будет доступ для чтения и записи к объявленным свойствам того NSObject, внутри которого реализован блок.
- Вы можете получать доступ к объявленным свойствам NSObject внутри независимых блоковых объектов, *только если* вы работаете с методами-установщиками и методами-получателями этих свойств. Вы не сможете получить доступ к объявленным свойствам объекта внутри независимого блокового объекта с помощью точечной нотации.

## Обсуждение

Сначала научимся работать с переменными, которые являются локальными для реализаций двух блоковых объектов. Один из этих блоковых объектов будет встраиваемым, а другой — независимым:

```
void (^independentBlockObject)(void) = ^(void){

  NSInteger localInteger = 10;

  NSLog(@"local integer = %ld", (long)localInteger);

  localInteger = 20;

  NSLog(@"local integer = %ld", (long)localInteger);

};
```

При активизации этого блокового объекта те значения, которые мы присваиваем, выводятся в окне консоли:

```
local integer = 10
local integer = 20
```

Пока все несложно. Теперь рассмотрим встраиваемые блоковые объекты и переменные, которые являются для них локальными:

```
- (void) simpleMethod{

  NSUInteger outsideVariable = 10;

  NSMutableArray *array = [[NSMutableArray alloc]
                           initWithObjects:@"obj1",
                           @"obj2", nil];

  [array sortUsingComparator:^NSComparisonResult(id obj1, id obj2) {
    NSUInteger insideVariable = 20;

    NSLog(@"Outside variable = %lu", (unsigned long)outsideVariable);
```

```
    NSLog(@"Inside variable = %lu", (unsigned long)insideVariable);

    /* Возвращаем значение для блокового объекта. */
    return NSOrderedSame;

  }];

}
```

Метод экземпляра sortUsingComparator:, относящийся к классу NSMutableArray, пытается сортировать изменяемый массив. Цель кода, приведенного в данном примере, — просто продемонстрировать использование локальных переменных. Можно и не задаваться тем, что именно делает этот метод.

Блоковый объект может считывать информацию и записывать данные в собственную локальную переменную `insideVariable`. При этом по умолчанию блоковый объект имеет доступ только для чтения к переменной `outsideVariable`. Чтобы блоковый объект мог записывать информацию в `outsideVariable`, нужно поставить перед `outsideVariable` префикс `__block`, указывающий соответствующий тип хранения:

```
- (void) simpleMethod{

  __block NSUInteger outsideVariable = 10;

  NSMutableArray *array = [[NSMutableArray alloc]
                           initWithObjects:@"obj1",
                           @"obj2", nil];


  [array sortUsingComparator:^NSComparisonResult(id obj1, id obj2) {

     NSUInteger insideVariable = 20;
     outsideVariable = 30;

     NSLog(@"Outside variable = %lu", (unsigned long)outsideVariable);
     NSLog(@"Inside variable = %lu", (unsigned long)insideVariable);

     /* Возвращаем значение для блокового объекта. */
     return NSOrderedSame;

  }];

}
```

Доступ к `self` во встраиваемых блоковых объектах не вызывает никаких проблем, пока `self` определяется в лексической области видимости, внутри которой создается встраиваемый блоковый объект. Например, в данной ситуации блоковый объект сможет получить доступ к `self`, поскольку метод `simpleMethod` является методом экземпляра класса языка Objective-C:

```
- (void) simpleMethod{

  NSMutableArray *array = [[NSMutableArray alloc]
                           initWithObjects:@"obj1",
                           @"obj2", nil];

  [array sortUsingComparator:^NSComparisonResult(id obj1, id obj2) {

    NSLog(@"self = %@", self);

    /* Возвращаем значение для блокового объекта. */
    return NSOrderedSame;

  }];

}
```

Не внеся изменений в реализацию вашего блокового объекта, вы не сможете получить доступ к `self` в независимом блоковом объекте. При попытке скомпилировать данный код мы получим ошибку времени компиляции:

```
void (^incorrectBlockObject)(void) = ^{
  NSLog(@"self = %@", self); /* self здесь не определен. */
};
```

Если вы хотите получить доступ к `self` в независимом блоковом объекте, просто передайте объект, представляемый `self`, вашему блоковому объекту в качестве параметра:

```
void (^correctBlockObject)(id) = ^(id self){

  NSLog(@"self = %@", self);

};

- (void) callCorrectBlockObject{

  correctBlockObject(self);

}
```

Этому параметру не обязательно присваивать имя self. Ему можно дать любое имя. Тем не менее если назвать этот параметр self, то можно будет просто собрать код блокового объекта позже и поместить его в реализацию метода на языке Objective-C. Не придется менять имя каждого экземпляра переменной на self, чтобы код был воспринят компилятором.

Рассмотрим объявленные свойства и посмотрим, как блоковые объекты могут получать к ним доступ. При работе со встраиваемыми блоковыми объектами можно применять точечную нотацию — она позволяет считывать информацию из объявленных свойств `self` или записывать в них данные. Допустим, например, что у нас в классе есть объявленное свойство типа `NSString`, которое называется `stringProperty`:

```
#import "AppDelegate.h"

@interface AppDelegate()
@property (nonatomic, copy) NSString *stringProperty;
@end

@implementation AppDelegate
```

Теперь не составляет труда получить доступ к этому свойству во встраиваемом блоковом объекте:

```
- (void) simpleMethod{

  NSMutableArray *array = [[NSMutableArray alloc]
                           initWithObjects:@"obj1",
                           @"obj2", nil];

  [array sortUsingComparator:^NSComparisonResult(id obj1, id obj2) {

    NSLog(@"self = %@", self);

    self.stringProperty = @"Block Objects";

    NSLog(@"String property = %@", self.stringProperty);

    /* Возвращаем значение для блокового объекта. */
    return NSOrderedSame;

  }];

}
```

Но в независимом блоковом объекте нельзя использовать точечную нотацию для считывания объявленного свойства или записи информации в это свойство:

```
void (^correctBlockObject)(id) = ^(id self){

  NSLog(@"self = %@", self);

  /* Вместо этого используем метод-установщик */
  self.stringProperty = @"Block Objects"; /* Ошибка времени компиляции */

  /* Вместо этого используем метод-получатель. */
  NSLog(@"self.stringProperty = %@",
        self.stringProperty); /* Ошибка времени компиляции */

};
```

В данном сценарии будем пользоваться методом-установщиком и методом-получателем синтезированного свойства:

```
void (^correctBlockObject)(id) = ^(id self){

  NSLog(@"self = %@", self);
```

```
  /* Это будет работать нормально. */
  [self setStringProperty:@"Block Objects"];

  /* Это также будет работать нормально. */
  NSLog(@"self.stringProperty = %@",
       [self stringProperty]);

};
```

Когда дело касается встраиваемых блоковых объектов, необходимо учитывать лишь одно *очень* важное правило: встраиваемые блоковые объекты копируют значения для переменных в своей лексической области видимости. Если вы не понимаете, что это значит, — не волнуйтесь. Рассмотрим пример:

```
typedef void (^BlockWithNoParams)(void);

- (void) scopeTest{

  NSUInteger integerValue = 10;

  BlockWithNoParams myBlock = ^{
    NSLog(@"Integer value inside the block = %lu",
          (unsigned long)integerValue);
  };

  integerValue = 20;

  /* Вызываем блок здесь после изменения
     значения переменной integerValue. */
  myBlock();

  NSLog(@"Integer value outside the block = %lu",
        (unsigned long)integerValue);

}
```

Мы определяем целочисленную локальную переменную и сначала присваиваем ей значение `10`. Затем реализуем блоковый объект, но *пока не вызываем* его. После того как блоковый объект *реализован*, мы просто изменяем значение локальной переменной, которую затем (после того как мы его вызовем) попытается считать блоковый объект. Сразу после изменения значения локальной переменной на `20` вызываем блоковый объект. Логично предположить, что блоковый объект выведет для переменной на консоль значение `20`, но этого не произойдет. Он выведет значение `10`, как показано здесь:

```
Integer value inside the block = 10
Integer value outside the block = 20
```

Вот что здесь происходит. Блоковый объект сохраняет для себя копию переменной `integerValue`, доступную только для чтения, и делает это именно там, где реализуется блок. Напрашивается вопрос: почему же блоковый объект принимает *доступное только для чтения* значение переменной `integerValue`? Ответ прост,

и мы уже дали его в этом разделе. Если у локальной переменной нет префикса __block, означающего соответствующий тип хранения, локальные переменные в лексической области видимости блокового объекта просто передаются блоковому объекту как переменные, доступные только для чтения. Следовательно, чтобы изменить это поведение, мы могли бы изменить реализацию метода scopeTest и сопроводить переменную integerValue префиксом __block, указывающим тип хранения. Это делается так:

```
- (void) scopeTest{

  __block NSUInteger integerValue = 10;


  BlockWithNoParams myBlock = ^{
    NSLog(@"Integer value inside the block = %lu",
          (unsigned long)integerValue);
  };

  integerValue = 20;

  /* Вызываем блок здесь после изменения
     значения переменной integerValue. */
  myBlock();

  NSLog(@"Integer value outside the block = %lu",
        (unsigned long)integerValue);

}
```

Теперь, если вывести на консоль результаты после вызова метода scopeTest, мы увидим следующее:

```
Integer value inside the block = 20
Integer value outside the block = 20
```

Итак, в данном разделе мы довольно подробно рассмотрели вопросы использования переменных с блоковыми объектами. Рекомендую вам написать несколько блоковых объектов и попытаться использовать в них переменные. Присваивайте им переменные, считывайте из них информацию, чтобы лучше разобраться с тем, как в блоковых объектах применяются переменные. Перечитайте этот раздел, если случайно забудете правила, регулирующие доступ к переменным в блоковых объектах.

# 7.3. Вызов блоковых объектов

## Постановка задачи

Вы научились создавать блоковые объекты, а теперь требуется их исполнять и получать определенные результаты.

## Решение

Исполняйте ваши блоковые объекты так же, как и функции на языке C. Подробнее об этом — в подразделе «Обсуждение».

## Обсуждение

В разделах 7.1 и 7.2 вы видели примеры вызова блоковых объектов. В данном разделе приводятся более конкретные примеры.

Если у вас есть независимый блоковый объект, его можно вызвать так же, как мы вызывали бы функцию на языке C:

```
void (^simpleBlock)(NSString *) = ^(NSString *paramString){
  /* Реализуем блоковый объект и используем параметр paramString. */
};

- (void) callSimpleBlock{
  simpleBlock(@"O'Reilly");

}
```

Если вы хотите вызвать независимый блоковый объект внутри другого независимого блокового объекта, действуйте так же, как при активизации метода на языке C:

```
NSString *(^trimString)(NSString *) = ^(NSString *inputString){

  NSString *result = [inputString stringByTrimmingCharactersInSet:
                      [NSCharacterSet whitespaceCharacterSet]];
  return result;

};

NSString *(^trimWithOtherBlock)(NSString *) = ^(NSString *inputString){
  return trimString(inputString);
};

- (void) callTrimBlock{

  NSString *trimmedString = trimWithOtherBlock(@" O'Reilly ");
  NSLog(@"Trimmed string = %@", trimmedString);

}
```

Продолжим данный пример и вызовем метод `callTrimBlock` на языке Objective-C:

```
[self callTrimBlock];
```

Метод `callTrimBlock` вызовет блоковый объект `trimWithOtherBlock`, а этот объект вызовет блоковый объект `trimString`, чтобы обрезать указанную строку. Отсечение строки — простая операция, для ее выполнения требуется всего одна строка кода. Но этот пример демонстрирует, как можно вызывать блоковые объекты внутри блоковых объектов.

## См. также

Разделы 7.1 и 7.2.

# 7.4. Решение с помощью GCD задач, связанных с пользовательским интерфейсом

## Постановка задачи

Интерфейс программирования приложений GCD используется для параллельного программирования, и необходимо узнать, каков оптимальный способ его применения с другими API, связанными с пользовательским интерфейсом.

## Решение

Воспользуйтесь функцией `dispatch_get_main_queue`.

## Обсуждение

Задачи, связанные с пользовательским интерфейсом, должны выполняться в главном потоке. Поэтому единственным каналом для передачи в GCD задач, связанных с пользовательским интерфейсом, и их выполнения оказывается главная очередь. В качестве описателя главной диспетчерской очереди можно применять функцию `dispatch_get_main_queue`.

Существует два способа направления задач в основную очередь. Оба этих способа асинхронны и позволяют не прерывать исполнения программы на время, пока завершается операция:

- *функция* `dispatch_async` выполняет блоковый объект применительно к диспетчерской очереди;
- *функция* `dispatch_async_f` выполняет функцию C применительно к диспетчерской очереди.

Метод dispatch_sync нельзя применять к главной очереди, поскольку он заблокирует поток на неопределенное время и ваше приложение войдет во взаимную блокировку. Все задачи, направляемые в GCD через главную очередь, должны туда направляться асинхронно.

Рассмотрим использование функции `dispatch_async`, которая принимает два параметра:

- *описатель диспетчерской очереди* — диспетчерская очередь, в которой должна выполняться задача;
- *блоковый объект* — блоковый объект, посылаемый в диспетчерскую очередь для асинхронного выполнения.

Рассмотрим пример. В операционной системе iOS следующий код будет выводить пользователю предупреждение, и при этом будет применяться главная очередь:

```
dispatch_queue_t mainQueue = dispatch_get_main_queue();

dispatch_async(mainQueue, ^(void) {

  [[[UIAlertView alloc] initWithTitle:@"GCD"
                              message:@"GCD is amazing!"
                             delegate:nil
                    cancelButtonTitle:@"OK"
                    otherButtonTitles:nil, nil] show];

});
```

Как видите, функция GCD dispatch_async не имеет ни параметров, ни возвращаемого значения. Блоковый объект, передаваемый данной функции для выполнения задачи, должен самостоятельно собирать данные для этого. В только что рассмотренном примере кода присутствует вид с предупреждением, имеющий все значения, которые требуются ему для выполнения задачи. Но так бывает не всегда. Иногда вам необходимо удостовериться, что блоковый объект, передаваемый GCD, располагает в собственной области видимости всеми значениями, которые требуются для решения стоящей перед ним задачи.

Если запустить данное приложение в эмуляторе iOS, пользователь увидит примерно такую картинку, как на рис. 7.1.

В общем-то, этот результат не слишком впечатляет. Так благодаря чему же главная очередь так интересна? Ответ прост: когда приходится задействовать GCD на полную мощность, например, чтобы выполнить сложные вычисления в параллельных или последовательных потоках, вам, возможно, понадобится отображать текущие результаты для пользователя или перемещать какой-либо компонент на экране. В таких случаях *необходимо* применять основную очередь, так как это задачи, связанные с пользовательским интерфейсом. Функции, рассмотренные в этом разделе, дают *единственную* возможность выйти из параллельной или последовательной очереди для обновления пользовательского интерфейса, не прекращая работу с GCD. Можете себе представить, насколько они важны.

Если вы не хотите передавать блоковый объект для выполнения в главную очередь, можно передать объект, представляющий собой функцию на языке C. Передавайте функции `dispatch_async_f` все те функции C, которые предполагается направлять на выполнение в GCD и которые относятся к работе пользовательского интерфейса. Мы можем получить такие же результаты, как на рис. 7.1, используя вместо блоковых объектов функции на языке C, при этом потребуется внести в код лишь незначительные корректировки.

Как было указано ранее, функция `dispatch_async_f` позволяет передавать указатель на контекст, определяемый приложением. Этот контекст впоследствии может использоваться вызываемой нами функцией C. Итак, создадим структуру, в которой будут содержаться значения — в частности, заголовок предупреждающего вида,


**Рис. 7.1.** Предупреждение, при выводе которого применялись асинхронные вызовы к GCD.

сообщение и надпись Cancel (Отмена) на соответствующей кнопке. Когда приложение запустится, мы поместим в эту структуру все значения и передадим ее функции C для отображения. Вот как определяется эта структура:

```
typedef struct{
  char *title;
  char *message;
  char *cancelButtonTitle;
} AlertViewData;
```

Итак, продолжим и реализуем функцию C, которая в дальнейшем будет вызываться с GCD. Эта функция должна ожидать параметр типа `void *`, тип которого затем приводится к `AlertViewData *`. Иными словами, мы ожидаем от вызывающей стороны этой функции, что нам будет передана ссылка на данные, необходимые для работы предупреждающего вида, которые инкапсулированы в структуре `AlertViewData`:

```
void displayAlertView(void *paramContext){

  AlertViewData *alertData = (AlertViewData *)paramContext;

  NSString *title =
    [NSString stringWithUTF8String:alertData->title];

  NSString *message =
    [NSString stringWithUTF8String:alertData->message];

  NSString *cancelButtonTitle =
    [NSString stringWithUTF8String:alertData->cancelButtonTitle];

  [[[UIAlertView alloc] initWithTitle:title
                              message:message
                             delegate:nil
                    cancelButtonTitle:cancelButtonTitle
                    otherButtonTitles:nil, nil] show];

  free(alertData);

}
```

Причина, по которой мы применяем free к переданному нам контексту именно здесь, а не на вызывающей стороне, заключается в том, что вызывающая сторона будет выполнять эту функцию C асинхронно и не сможет узнать, когда выполнение функции на языке C завершится. Поэтому вызывающая сторона должна выделить достаточный объем памяти для контекста AlertViewData (операция malloc), и функция C displayAlertView должна высвободить это пространство.

А теперь вызовем функцию `displayAlertView` применительно к основной очереди и передадим ей контекст (структуру, содержащую данные для предупреждающего вида):

```
- (BOOL)            application:(UIApplication *)application
  didFinishLaunchingWithOptions:(NSDictionary *)launchOptions{

  dispatch_queue_t mainQueue = dispatch_get_main_queue();

  AlertViewData *context = (AlertViewData *)
  malloc(sizeof(AlertViewData));

  if (context != NULL){
    context->title = "GCD";
    context->message = "GCD is amazing.";
    context->cancelButtonTitle = "OK";

    dispatch_async_f(mainQueue,
                     (void *)context,
                     displayAlertView);
```

```
  }

  self.window = [[UIWindow alloc] initWithFrame:
                 [[UIScreen mainScreen] bounds]];

  self.window.backgroundColor = [UIColor whiteColor];
  [self.window makeKeyAndVisible];
  return YES;
}
```

Если активизировать метод класса `currentThread`, относящийся к классу `NSThread`, то выяснится, что блоковые объекты или функции C, направляемые вами в главную очередь, действительно работают в главном потоке:

```
- (BOOL)            application:(UIApplication *)application
  didFinishLaunchingWithOptions:(NSDictionary *)launchOptions{

  dispatch_queue_t mainQueue = dispatch_get_main_queue();

  dispatch_async(mainQueue, ^(void) {
    NSLog(@"Current thread = %@", [NSThread currentThread]);
    NSLog(@"Main thread = %@", [NSThread mainThread]);
  });

  self.window = [[UIWindow alloc] initWithFrame:
                 [[UIScreen mainScreen] bounds]];
  self.window.backgroundColor = [UIColor whiteColor];
  [self.window makeKeyAndVisible];
  return YES;
}
```

Вывод данного кода будет примерно таким:

```
Current thread = <NSThread: 0x4b0e4e0>{name = (null), num = 1}
Main thread = <NSThread: 0x4b0e4e0>{name = (null), num = 1}
```

Итак, мы изучили, как с помощью GCD решаются задачи, связанные с пользовательским интерфейсом. Перейдем к другим темам — в частности, поговорим о том, как выполнять задачи параллельно, используя параллельные очереди (см. разделы 7.5 и 7.6), и как при необходимости смешивать создаваемый код с кодом пользовательского интерфейса.

# 7.5. Синхронное решение с помощью GCD задач, не связанных с пользовательским интерфейсом

## Постановка задачи

Необходимо выполнять синхронные задачи, в которых не участвует код, связанный с пользовательским интерфейсом.

## Решение

Воспользуйтесь функцией `dispatch_sync`.

## Обсуждение

Иногда необходимо решать задачи, никак не связанные с пользовательским интерфейсом, либо осуществлять процессы, которые взаимодействуют и с пользовательским интерфейсом, но в то же время заняты решением долговременных задач. Например, вам может понадобиться загрузить изображение, а после загрузки отобразить его для пользователя. Процесс загрузки совершенно не связан с пользовательским интерфейсом.

Любая задача, не связанная с пользовательским интерфейсом, позволяет применять глобальные параллельные очереди, которые предоставляет GCD. Они могут выполняться как синхронно, так и асинхронно. Но синхронное выполнение *не означает*, что программа дожидается, пока выполнится определенный фрагмент кода, а потом продолжает работу. Это лишь означает, что параллельная очередь дождется выполнения вашей задачи и только потом перейдет к выполнению следующего блока кода, стоящего в очереди. Когда вы ставите в параллельную очередь блоковый объект, ваша собственная программа *всегда* продолжает работу, не дожидаясь, пока выполнится код, стоящий в очереди. Дело в том, что параллельные очереди (как понятно из их названия) выполняют свой код в неглавных потоках. (Из этого правила есть исключение: когда задача передается в параллельную или последовательную очередь посредством функции `dispatch_sync`, система iOS при наличии такой возможности запускает задачу в *текущем* потоке. А это *может быть* и главный поток в зависимости от актуальной ветви кода. Это специальная оптимизация, запрограммированная в GCD, и вскоре мы обсудим ее подробнее.)

Если вы отправляете синхронную задачу в параллельную очередь и в то же время отправляете синхронную задачу в *другую* параллельную очередь, то две эти синхронные задачи будут выполняться асинхронно друг относительно друга, так как относятся к *двум разным параллельным очередям*. Этот нюанс важно понимать, поскольку иногда необходимо гарантировать, что задача B начнет выполняться только после того, как завершится задача A. Чтобы обеспечить такую последовательность, эти две задачи нужно синхронно отправлять в *одну и ту же* очередь.

Синхронные задачи в диспетчерской очереди можно выполнять с помощью функции `dispatch_sync`. Все, что от вас требуется, — снабдить эту функцию описателем той очереди, в которой должна выполняться задача, а также блоком кода, который должен выполниться в данной очереди.

Рассмотрим пример. Данная функция выводит на консоль числа от 1 до 1000, всю последовательность подряд, и при этом не блокирует основной поток. Мы можем создать блоковый объект, выполняющий подсчет за нас, и синхронно (дважды) вызвать этот же блоковый объект:

```
void (^printFrom1To1000)(void) = ^{

  NSUInteger counter = 0;
  for (counter = 1;
```

```
        counter <= 1000;
        counter++){

    NSLog(@"Counter = %lu - Thread = %@",
          (unsigned long)counter,
          [NSThread currentThread]);

  }

};
```

Итак, попробуем активизировать этот блоковый объект с помощью GCD:

```
- (BOOL)            application:(UIApplication *)application
  didFinishLaunchingWithOptions:(NSDictionary *)launchOptions{
  dispatch_queue_t concurrentQueue =
  dispatch_get_global_queue(DISPATCH_QUEUE_PRIORITY_DEFAULT, 0);

  dispatch_sync(concurrentQueue, printFrom1To1000);
  dispatch_sync(concurrentQueue, printFrom1To1000);
  // Точка переопределения для дополнительной настройки
  // после запуска приложения
  [self.window makeKeyAndVisible];
  return YES;
}
```

Запустив этот код, вы заметите, что счетчик работает в главном потоке даже при том, что вы поставили эту задачу на выполнение в параллельную очередь. Оказывается, что это явление — специальная оптимизация GCD. Функция `dispatch_sync` будет использовать актуальный поток, то есть поток, который вы задействуете при направлении задачи в очередь, всякий раз, когда это возможно. В этом и заключается упомянутая оптимизация. Вот что об этом пишет Apple в справке по GCD: *«В целях оптимизации работы данная функция активизирует блок кода в актуальном потоке всякий раз, когда это возможно»*.

Чтобы выполнить вместо блокового объекта функцию на языке C и сделать это синхронно, в диспетчерской очереди, используйте функцию `dispatch_sync_f`. Давайте просто преобразуем код, написанный для блокового объекта `printFrom1To1000`, в эквивалентную ему функцию на языке C:

```
void printFrom1To1000(void *paramContext){

  NSUInteger counter = 0;
  for (counter = 1;
       counter <= 1000;
       counter++){

    NSLog(@"Counter = %lu - Thread = %@",
          (unsigned long)counter,
          [NSThread currentThread]);

  }

}
```

А теперь можно воспользоваться функцией dispatch_sync_f для выполнения функции printFrom1To1000 в параллельной очереди:

```
- (BOOL) application:(UIApplication *)application
didFinishLaunchingWithOptions:(NSDictionary *)launchOptions{
dispatch_queue_t concurrentQueue =
  dispatch_get_global_queue(DISPATCH_QUEUE_PRIORITY_DEFAULT, 0);

dispatch_sync_f(concurrentQueue,
                NULL,
                printFrom1To1000);

dispatch_sync_f(concurrentQueue,
                NULL,
                printFrom1To1000);
self.window = [[UIWindow alloc]
initWithFrame:[[UIScreen mainScreen] bounds]];
self.window.backgroundColor = [UIColor whiteColor];
[self.window makeKeyAndVisible];
return YES;
}
```

Первый параметр функции dispatch_get_global_queue указывает приоритет параллельной очереди. Этот показатель GCD должен получить и предоставить программисту. Чем выше приоритет, тем больше квантов процессорного времени будет уделяться коду, выполняемому в этой очереди. В качестве первого параметра функции dispatch_get_global_queue можно использовать любое из следующих значений:

- DISPATCH_QUEUE_PRIORITY_LOW — ваша задача будет получать меньше процессорного времени, чем выделяется на задачу в среднем;
- DISPATCH_QUEUE_PRIORITY_DEFAULT — ваша задача получит стандартный системный приоритет;
- DISPATCH_QUEUE_PRIORITY_HIGH — ваша задача будет получать больше процессорного времени, чем выделяется на задачу в среднем.

Второй параметр функции dispatch_get_global_queue зарезервирован, ему всегда следует передавать значение 0.

В данном разделе было рассмотрено, как передавать задачи в параллельные очереди для синхронного исполнения. В следующем разделе обсудим асинхронное исполнение в параллельных очередях, а в разделе 7.10 будет показано, как исполнять задачи синхронно и асинхронно в последовательных очередях, создаваемых вами для приложений.

## См. также

Разделы 7.6 и 7.10.

# 7.6. Асинхронное решение с помощью GCD задач, не связанных с пользовательским интерфейсом

## Постановка задачи

Необходимо иметь возможность решать задачи, не связанные с пользовательским интерфейсом, с помощью GCD.

## Решение

Вот здесь GCD и проявляется во всей красе: при асинхронном выполнении блоков кода в главной очереди, последовательных и параллельных очередях. Не сомневаюсь, что, дочитав этот раздел, вы будете совершенно убеждены в том, что будущее многопоточных приложений неразрывно связано с GCD, который в перспективе заменит потоки, применяемые в современных программах.

Чтобы выполнять в диспетчерской очереди асинхронные задачи, следует пользоваться одной из следующих функций:

- `dispatch_async` — отправляет блоковый объект в диспетчерскую очередь (и объект и очередь указываются в соответствующих параметрах) для асинхронного выполнения;
- `dispatch_async_f` — отправляет в диспетчерскую очередь функцию языка C вместе со ссылкой на контекст (все три элемента указываются в соответствующих параметрах) для асинхронного выполнения.

## Обсуждение

Рассмотрим реальный пример. Напишем приложение для iOS, которое позволит нам скачивать изображение из Интернета по имеющейся гиперссылке (URL). После завершения загрузки наша программа должна отобразить изображение для пользователя. Далее приведен план работы и описано, как будут применены те или иные концепции, связанные с GCD, которые мы уже успели изучить.

1. Мы собираемся асинхронно запускать блоковый объект в параллельной очереди.
2. В ходе выполнения этого блока будем однократно (синхронно) запускать другой блоковый объект. Его мы будем использовать для скачивания изображения по URL, при этом будет применяться функция `dispatch_sync`. Мы поступаем именно так, поскольку хотим, чтобы обработка остального кода, стоящего в данной параллельной очереди, не начиналась, пока не загрузится изображение. В результате мы заставляем подождать только одну параллельную очередь, а не все остальные очереди. Если синхронно скачивать файл по URL из асинхронного блока кода, мы заблокируем лишь очередь, обрабатывающую синхронную функцию, но не главный поток. Вся операция так и остается асинхронной с точ-

ки зрения главного потока. Мы решаем основную задачу: при загрузке изображения главный поток не блокируется.

3. Сразу после того, как загрузка изображения завершится, мы синхронно выполним блоковый объект в главной очереди (см. раздел 7.4), чтобы отобразить картинку в пользовательском интерфейсе.

   Каркас для планируемой программы совершенно прост:

```
- (void) viewDidAppear:(BOOL)animated{
  [super viewDidAppear:animated];
  dispatch_queue_t concurrentQueue =
  dispatch_get_global_queue(DISPATCH_QUEUE_PRIORITY_DEFAULT, 0);

  dispatch_async(concurrentQueue, ^{

    __block UIImage *image = nil;

    dispatch_sync(concurrentQueue, ^{
      /* Здесь скачивается изображение. */
    });

    dispatch_sync(dispatch_get_main_queue(), ^{
      /* Здесь мы демонстрируем изображение пользователю и делаем это
         в главной очереди. */
    });

  });
}
```

Второй вызов к `dispatch_sync`, после которого отобразится картинка, будет выполняться в очереди после первого синхронного вызова, который обеспечивает загрузку изображения. Именно этого мы и добивались, поскольку нам *необходимо* дождаться, пока изображение загрузится полностью, и только после этого мы сможем отобразить его для пользователя. Итак, после завершения скачивания изображения мы выполняем второй блоковый объект, но на этот раз — в главной очереди.

Скачаем изображение и отобразим его для пользователя. Это мы сделаем в методе экземпляра `viewDidAppear:`, относящемся к контроллеру вида, который в данный момент отображается в приложении для iPhone:

```
- (void) viewDidAppear:(BOOL)paramAnimated{
  [super viewDidAppear:paramAnimated];

  dispatch_queue_t concurrentQueue =
  dispatch_get_global_queue(DISPATCH_QUEUE_PRIORITY_DEFAULT, 0);

  dispatch_async(concurrentQueue, ^{

    __block UIImage *image = nil;

    dispatch_sync(concurrentQueue, ^{
```

```
    /* Здесь скачивается изображение. */

   /* Изображение iPad с сайта Apple. Гиперссылка слишком длинная,
      поэтому ее нужно правильно разбить на две строки. */
    NSString *urlAsString = @"http://images.apple.com/mobileme/features"\
                            "/images/ipad_findyouripad_20100518.jpg";

    NSURL *url = [NSURL URLWithString:urlAsString];

    NSURLRequest *urlRequest = [NSURLRequest requestWithURL:url];

    NSError *downloadError = nil;
    NSData *imageData = [NSURLConnection
                         sendSynchronousRequest:urlRequest
                         returningResponse:nil
                         error:&downloadError];

    if (downloadError == nil &&
        imageData != nil){

      image = [UIImage imageWithData:imageData];
      /* Изображение у нас есть. Теперь можно его использовать. */

    }
    else if (downloadError != nil){
      NSLog(@"Error happened = %@", downloadError);
    } else {
      NSLog(@"No data could get downloaded from the URL.");
    }

  });

  dispatch_sync(dispatch_get_main_queue(), ^{
    /* Здесь картинка отображается, и это происходит в главной очереди. */

    if (image != nil){
      /* Здесь создается вид с изображением. */
      UIImageView *imageView = [[UIImageView alloc]
                                initWithFrame:self.view.bounds];

    /* Задаем характеристики изображения. */
    [imageView setImage:image];

    /* Убеждаемся, что изображение масштабировано правильно. */
    [imageView setContentMode:UIViewContentModeScaleAspectFit];

    /* Добавляем изображение к виду данного контроллера вида. */
    [self.view addSubview:imageView];

    } else {
      NSLog(@"Image isn't downloaded. Nothing to display.");
```

```
        }
    });
});
}
```

Как показано на рис. 7.2, мы успешно загрузили изображение, а также создали вид изображения, в котором картинка будет представлена пользователю в графическом интерфейсе.


**Рис. 7.2.** Загрузка изображения и демонстрация его пользователю, применяется GCD

Приведем другой пример. Допустим, у нас есть массив из 10 000 случайных чисел, которые сохранены в файле на диске. Мы хотим загрузить этот файл в память и отсортировать числа в порядке возрастания (то есть сделать так, чтобы список начинался с наименьшего числа). Потом мы хотим отобразить полученный список

для пользователя. Инструмент управления, который будет применяться при этой операции, определяется тем, для какой системы вы пишете программу. В случае с iOS идеальным выходом было бы использовать экземпляр `UITableView`, а при работе с Mac OS X — экземпляр `NSTableView`. Поскольку массива у нас еще нет, начнем с его создания, потом загрузим этот массив, а потом отобразим.

Вот два метода, которые помогут нам найти место на диске устройства, где мы собираемся сохранить массив из 10 000 случайных чисел:

```
- (NSString *) fileLocation{

  /* Получаем каталог (-и) документа. */
  NSArray *folders =
  NSSearchPathForDirectoriesInDomains(NSDocumentDirectory,
                                      NSUserDomainMask,
                                      YES);

  /* Мы что-нибудь нашли? */
  if ([folders count] == 0){
    return nil;
  }

  /* Получаем первый каталог. */
  NSString *documentsFolder = [folders objectAtIndex:0];

  /* Прикрепляем имя файла к концу пути документа. */
  return [documentsFolder
          stringByAppendingPathComponent:@"list.txt"];

}

- (BOOL) hasFileAlreadyBeenCreated{

  BOOL result = NO;

  NSFileManager *fileManager = [[NSFileManager alloc] init];
  if ([fileManager fileExistsAtPath:[self fileLocation]]){
    result = YES;
  }

  return result;
}
```

А вот теперь очень важный нюанс. Мы хотим сохранить на диске массив из 10 000 случайных чисел, *если, и только если* мы не создавали такой массив на диске раньше. В противном случае мы сразу загрузим массив с диска. Если же прежде мы не создавали этот массив на диске, то сначала создадим его, а потом перейдем к загрузке массива с диска. В итоге, если считывание массива с диска пройдет успешно, мы отсортируем этот массив в порядке возрастания и, наконец, отобразим результаты для пользователя в графическом интерфейсе. Реализацию отображения результатов пользователю оставляю вам для самостоятельной работы.

```
- (void) viewDidAppear:(BOOL)paramAnimated{

  [super viewDidAppear:paramAnimated];

  dispatch_queue_t concurrentQueue =
  dispatch_get_global_queue(DISPATCH_QUEUE_PRIORITY_DEFAULT, 0);

  /* Если мы еще не отсортировали массив из 10000 случайных чисел
   на диске ранее, сгенерируем эти числа сейчас, а потом сохраним
   их на диск в массиве. */
  dispatch_async(concurrentQueue, ^{

    NSUInteger numberOfValuesRequired = 10000;

    if ([self hasFileAlreadyBeenCreated] == NO){
      dispatch_sync(concurrentQueue, ^{

        NSMutableArray *arrayOfRandomNumbers =
        [[NSMutableArray alloc] initWithCapacity:numberOfValuesRequired];

        NSUInteger counter = 0;
        for (counter = 0;
             counter < numberOfValuesRequired;
             counter++){
          unsigned int randomNumber =
          arc4random() % ((unsigned int)RAND_MAX + 1);
          [arrayOfRandomNumbers addObject:
          [NSNumber numberWithUnsignedInt:randomNumber]];
        }

        /* Теперь записываем массив на диск. */
        [arrayOfRandomNumbers writeToFile:[self fileLocation]
                               atomically:YES];

      });
    }

    __block NSMutableArray *randomNumbers = nil;

    /* Считываем числа с диска и сортируем их в порядке возрастания. */
    dispatch_sync(concurrentQueue, ^{

      /* Если файл на данный момент уже создан, занимаемся его считыванием. */
      if ([self hasFileAlreadyBeenCreated]){
        randomNumbers = [[NSMutableArray alloc]
                         initWithContentsOfFile:[self fileLocation]];

        /* Теперь сортируем числа. */
        [randomNumbers sortUsingComparator:
         ^NSComparisonResult(id obj1, id obj2) {
```

```
            NSNumber *number1 = (NSNumber *)obj1;
            NSNumber *number2 = (NSNumber *)obj2;
            return [number1 compare:number2];

          }];
        }
      });

      dispatch_async(dispatch_get_main_queue(), ^{
        if ([randomNumbers count] > 0){
          /* Обновляем пользовательский интерфейс, задействуя числа
           из массива randomNumbers. */
        }
      });

  });
}
```

Функционал GCD далеко не ограничивается синхронным или асинхронным выполнением блоков кода или функций. В разделе 7.9 вы научитесь группировать блоковые объекты и готовить их к выполнению в диспетчерской очереди. Кроме того, рекомендую вам изучить разделы 7.7. и 7.8, где говорится о прочих функциях, которые предоставляются программисту в GCD.

## См. также

Разделы 7.4, 7.7 и 7.8.

# 7.7. Выполнение задач после задержки с помощью GCD

## Постановка задачи

Требуется выполнить код, но после определенной задержки. Задержку планируется указывать с помощью GCD.

## Решение

Воспользуйтесь функциями `dispatch_after` и `dispatch_after_f`.

## Обсуждение

Работая с фреймворком Core Foundation, можно активизировать селектор в объекте по истечении заданного временного промежутка с помощью метода `performSelector:withObject:afterDelay:`, относящегося к классу `NSObject`. Например:

```
- (void) printString:(NSString *)paramString{
  NSLog(@"%@", paramString);
}

- (BOOL)            application:(UIApplication *)application
  didFinishLaunchingWithOptions:(NSDictionary *)launchOptions{

  [self performSelector:@selector(printString:)
             withObject:@"Grand Central Dispatch"
             afterDelay:3.0];

  self.window = [[UIWindow alloc] initWithFrame:
                 [[UIScreen mainScreen] bounds]];

  // Точка переопределения для настройки после запуска приложения
  self.window.backgroundColor = [UIColor whiteColor];
  [self.window makeKeyAndVisible];
  return YES;
}
```

В данном примере мы приказываем среде времени исполнения вызвать метод `printString:` после трехсекундной задержки. Ту же операцию можно осуществить и в GCD с помощью функций `dispatch_after` и `dispatch_after_f`. Обе эти функции описаны далее.

- `dispatch_after` — направляет блоковый объект в диспетчерскую очередь по истечении заданного периода времени, указываемого в наносекундах. Эта функция требует следующих параметров:
  - *задержка в наносекундах* — количество наносекунд, в течение которых длится ожидание в определенной диспетчерской очереди в GCD (указываемой во втором параметре), после чего выполняется блоковый объект (задаваемый в третьем параметре);
  - *диспетчерская очередь* — диспетчерская очередь, в которой должен быть выполнен блоковый объект (указываемый в третьем параметре) после определенной задержки (задаваемой в первом параметре);
  - *блоковый объект* — блоковый объект, который должен быть активизирован в заданной диспетчерской очереди по истечении заданного количества наносекунд. Блоковый объект не должен иметь возвращаемого значения и не должен принимать никаких параметров (см. раздел 7.1).
- `dispatch_after_f` — направляет функцию на языке C в GCD для выполнения по истечении указанного периода времени, задаваемого в наносекундах. Данная функция принимает четыре параметра:
  - *задержка в наносекундах* — количество наносекунд, в течение которых длится ожидание в определенной диспетчерской очереди в GCD (указываемой во втором параметре), после чего выполняется заданная функция (задаваемая в четвертом параметре);
  - *диспетчерская очередь* — диспетчерская очередь, в которой должна быть выполнена функция на языке C (указываемая во втором параметре) после определенной задержки (задаваемой в первом параметре);

- *контекст* — адрес в памяти, по которому находится определенное значение, относящееся к неупорядоченному массиву данных (куче). Это значение должно передаваться функции C. Подробнее об этом говорилось в разделе 7.4;
- *функция на языке C* — адрес функции на языке C, которая должна быть выполнена по истечении определенного периода времени (указываемого в первом параметре) в заданной диспетчерской очереди (указываемой во втором параметре).

Хотя задержки рассчитываются в наносекундах, размерность задержки при диспетчеризации определяется самой системой iOS, и эта величина может быть менее точной, чем та, которую вы указываете в наносекундах.

Сначала рассмотрим пример работ с функцией `dispatch_after`:

```
- (BOOL) application:(UIApplication *)application
  didFinishLaunchingWithOptions:(NSDictionary *)launchOptions{

    double delayInSeconds = 2.0;

    dispatch_time_t delayInNanoSeconds =
    dispatch_time(DISPATCH_TIME_NOW, delayInSeconds * NSEC_PER_SEC);

    dispatch_queue_t concurrentQueue =
    dispatch_get_global_queue(DISPATCH_QUEUE_PRIORITY_DEFAULT, 0);

    dispatch_after(delayInNanoSeconds, concurrentQueue, ^(void){
    /* Здесь выполняются требуемые операции. */
  });
  // Точка переопределения для настройки после запуска приложения
  self.window.backgroundColor = [UIColor whiteColor];
  [self.window makeKeyAndVisible];
  return YES;
}
```

Как видите, параметр задержки в наносекундах для функций `dispatch_after` и `dispatch_after_f` должен относиться к типу `dispatch_time_t`, который является абстрактным представлением абсолютного времени. Чтобы получить значение этого параметра, можно пользоваться функцией `dispatch_time` так, как показано в данном образце кода. Вот параметры, которые можно сообщать функции `dispatch_time`.

- *Исходное время* — если обозначить этот параметр через `B`, а приращение времени (Delta Parameter) — через `D`, то результирующее время от этой функции будет равно `B+D`. Для этого параметра можно задать значение `DISPATCH_TIME_NOW`, определив таким образом в качестве базового времени *настоящий момент*, а потом указать приращение, добавляемое к этому времени, используя дельта-параметр.

- *Приращение, добавляемое к базовому времени,* — этот параметр дает количество наносекунд, добавляемых к параметру исходного времени для получения результата данной функции.

Например, чтобы задать временной промежуток 3 с начиная от настоящего момента, можно написать следующий код:

```
dispatch_time_t delay =
dispatch_time(DISPATCH_TIME_NOW, 3.0f * NSEC_PER_SEC);
```

А вот так задается период 0,5 с от настоящего момента:

```
dispatch_time_t delay =
dispatch_time(DISPATCH_TIME_NOW, (1.0 / 2.0f) * NSEC_PER_SEC);
```

Теперь рассмотрим, как можно использовать функцию `dispatch_after_f`:

```
void processSomething(void *paramContext){
  /* Здесь происходит обработка. */
  NSLog(@"Processing...");
}

- (BOOL)            application:(UIApplication *)application
  didFinishLaunchingWithOptions:(NSDictionary *)launchOptions{

    double delayInSeconds = 2.0;

    dispatch_time_t delayInNanoSeconds =
    dispatch_time(DISPATCH_TIME_NOW, delayInSeconds * NSEC_PER_SEC);

    dispatch_queue_t concurrentQueue =
    dispatch_get_global_queue(DISPATCH_QUEUE_PRIORITY_DEFAULT, 0);

    dispatch_after_f(delayInNanoSeconds,
                     concurrentQueue,
                     NULL,
                     processSomething);

    self.window = [[UIWindow alloc] initWithFrame:
                   [[UIScreen mainScreen] bounds]];

    // Точка переопределения для настройки после запуска приложения
    self.window.backgroundColor = [UIColor whiteColor];
    [self.window makeKeyAndVisible];
    return YES;
}
```

## См. также

Разделы 7.1 и 7.4.

# 7.8. Однократное выполнение задач с помощью GCD

## Постановка задачи

Необходимо убедиться в том, что определенный фрагмент кода выполняется один раз за весь жизненный цикл приложения, даже если он вызывается неоднократно из разных точек программы (в качестве примера можно привести инициализацию синглтона).

## Решение

Воспользуйтесь функцией `dispatch_once`.

## Обсуждение

Выделение и инициализация синглтона — одна из таких задач, которые должны происходить один, и только один раз за весь жизненный цикл приложения. Уверен, что вы можете вспомнить и другие аналогичные сценарии.

GCD позволяет указывать идентификатор для фрагмента кода при попытке выполнить этот код. Если GCD обнаруживает, что данный идентификатор уже передавался фреймворку ранее, то система не будет вновь выполнять этот блок кода. Функция, которая обеспечивает выполнение подобных задач, называется `dispatch_once`. Она может принимать два параметра.

- *Маркер* — маркер типа `dispatch_once_t`, содержащий сгенерированную GCD метку при первом выполнении блока кода. Если вы хотите, чтобы блок кода был выполнен лишь один раз, нужно указывать для данного метода один и тот же маркер независимо от того, когда он активизируется в приложении. Такой пример мы вскоре рассмотрим.
- *Блоковый объект* — блоковый объект, выполняемый не более одного раза. Блоковый объект не возвращает никаких значений и не принимает никаких параметров.

dispatch_once всегда выполняет свою задачу в актуальной очереди, используемой кодом, который делает вызов. Это может быть как последовательная, так и параллельная или главная очереди.

Например:

```
static dispatch_once_t onceToken;

void (^executedOnlyOnce)(void) = ^{

  static NSUInteger numberOfEntries = 0;
  numberOfEntries++;
```

```
  NSLog(@"Executed %lu time(s)", (unsigned long)numberOfEntries);

};

- (BOOL)            application:(UIApplication *)application
  didFinishLaunchingWithOptions:(NSDictionary *)launchOptions{

  dispatch_queue_t concurrentQueue =
    dispatch_get_global_queue(DISPATCH_QUEUE_PRIORITY_DEFAULT, 0);

  dispatch_once(&onceToken, ^{
    dispatch_async(concurrentQueue,
                   executedOnlyOnce);
  });

  dispatch_once(&onceToken, ^{
    dispatch_async(concurrentQueue,
                   executedOnlyOnce);
  });

  self.window = [[UIWindow alloc] initWithFrame:
                 [[UIScreen mainScreen] bounds]];

  self.window.backgroundColor = [UIColor whiteColor];
  [self.window makeKeyAndVisible];
  return YES;

}
```

Как видите, мы пытаемся активизировать блоковый объект `executedOnlyOnce` дважды с помощью функции `dispatch_once`, но на самом деле GCD выполняет этот блоковый объект лишь однажды, поскольку идентификатор, передаваемый функции `dispatch_once`, оба раза один и тот же.

В руководстве Cocoa Fundamentals Guide (Руководство по основам Cocoa) (https://developer.apple.com/library/ios/#documentation/General/Conceptual/DevPedia-CocoaCore/Singleton.html) Apple объясняется, как создавать синглтон. Исходный код довольно старый *и еще не обновлен* с учетом использования GCD и автоматического подсчета ссылок. Мы можем изменить эту модель, чтобы можно было пользоваться GCD и функцией `dispatch_once`. В результате мы сможем создавать совместно используемый экземпляр объекта:

```
#import "MySingleton.h"

@implementation MySingleton

- (instancetype) sharedInstance{
  static MySingleton *SharedInstance = nil;
  static dispatch_once_t onceToken;
  dispatch_once(&onceToken, ^{
    SharedInstance = [MySingleton new];
```

```
  });
  return SharedInstance;
}

@end
```

# 7.9. Объединение задач в группы с помощью GCD

## Постановка задачи

Требуется объединять блоки кода в группы и гарантировать, что GCD будет выполнять все задачи одну за другой, выстраивая таким образом зависимости между ними.

## Решение

Для создания групп в GCD пользуйтесь функцией `dispatch_group_create`.

## Обсуждение

GCD позволяет создавать *группы*. Пользуясь группами, можно поместить несколько задач в одном месте, выполнить их все, а по завершении работы получить об этом уведомление от GCD. Такая технология имеет большое прикладное значение. Допустим, например, что у вас есть приложение с пользовательским интерфейсом и вы хотите перезагрузить его компоненты в этом пользовательском интерфейсе. В пользовательском интерфейсе у вас имеется табличный вид, прокручиваемый вид и вид с изображением. Вы хотите перезагрузить содержимое этих компонентов с помощью следующих методов:

```
- (void) reloadTableView{
  /* Здесь перезагружается табличный вид. */
  NSLog(@"%s", __FUNCTION__);
}

- (void) reloadScrollView{
  /* Здесь выполняется работа. */
  NSLog(@"%s", __FUNCTION__);
}

- (void) reloadImageView{
  /* Здесь перезагружается вид с изображением. */
  NSLog(@"%s", __FUNCTION__);
}
```

На данный момент эти методы пусты, но вы можете позже поместить в них важный код, связанный с пользовательским интерфейсом. Сейчас мы собираемся

вызвать эти три метода один за другим и узнать, когда GCD закончит вызывать эти методы, в результате чего мы отобразим соответствующее сообщение для пользователя. Для этого нам придется воспользоваться группой. При работе с группами в GCD необходимо иметь представление о трех функциях:

- `dispatch_group_create` — создает описатель группы;
- `dispatch_group_async` — отправляет блок кода в группу для выполнения. Необходимо указать диспетчерскую очередь, в которой должен выполняться этот блок кода, *а также* группу, к которой этот блок кода относится;
- `dispatch_group_notify` — позволяет отправить блоковый объект, который необходимо выполнить после того, как все задачи, направленные в группу для выполнения, закончат свою работу. Эта функция также позволяет указывать диспетчерскую очередь, в которой должен выполняться данный блоковый объект.

Рассмотрим пример. Как объяснялось ранее, в этом примере мы собираемся активизировать методы `reloadTableView`, `reloadScrollView` и `reloadImageView` один за другим, а потом отобразить для пользователя сообщение о том, что задача выполнена. Для достижения этой цели применим мощные групповые функции, присущие GCD:

```
- (BOOL) application:(UIApplication *)application
  didFinishLaunchingWithOptions:(NSDictionary *)launchOptions{
    dispatch_group_t taskGroup = dispatch_group_create();
    dispatch_queue_t mainQueue = dispatch_get_main_queue();

    /* Перезагружаем табличный вид в главной очереди. */
    dispatch_group_async(taskGroup, mainQueue, ^{
      [self reloadTableView];
    });

    /* Перезагружаем прокручиваемый вид в главной очереди. */
    dispatch_group_async(taskGroup, mainQueue, ^{
      [self reloadScrollView];
    });

    /* Перезагружаем вид с изображением в главной очереди. */
    dispatch_group_async(taskGroup, mainQueue, ^{
      [self reloadImageView];
    });

    /* Когда все это будет сделано, диспетчеризуем следующий блок. */
    dispatch_group_notify(taskGroup, mainQueue, ^{
      /* Здесь происходит обработка. */
      [[[UIAlertView alloc] initWithTitle:@"Finished"
                                  message:@"All tasks are finished"
                                 delegate:nil
                        cancelButtonTitle:@"OK"
                        otherButtonTitles:nil, nil] show];

    });
```

```
    self.window = [[UIWindow alloc] initWithFrame:
                   [[UIScreen mainScreen] bounds]];
    self.window.backgroundColor = [UIColor whiteColor];
    [self.window makeKeyAndVisible];
    return YES;
}
```

Кроме работы с функцией `dispatch_group_async`, можно также направлять асинхронные функции на языке C, используя функцию `dispatch_group_async_f`.

GCDAppDelegate — это просто имя класса, из которого взят пример. Данное имя класса мы будем использовать для приведения типа контекстного объекта так, чтобы компилятор понимал наши команды.

Вот так:

```
void reloadAllComponents(void *context){

AppDelegate *self = (__bridge AppDelegate *)context;
  [self reloadTableView];
  [self reloadScrollView];
  [self reloadImageView];

}

- (BOOL)              application:(UIApplication *)application
  didFinishLaunchingWithOptions:(NSDictionary *)launchOptions{

    dispatch_group_t taskGroup =  dispatch_group_create();
    dispatch_queue_t mainQueue =  dispatch_get_main_queue();

    dispatch_group_async_f(taskGroup,
                           mainQueue,
                           (__bridge void *)self,
                           reloadAllComponents);

    /* Когда все это будет сделано, диспетчеризуем следующий блок. */
    dispatch_group_notify(taskGroup, mainQueue, ^{
      /* Здесь происходит обработка. */
      [[[UIAlertView alloc] initWithTitle:@"Finished"
                                  message:@"All tasks are finished"
                                 delegate:nil
                        cancelButtonTitle:@"OK"
                        otherButtonTitles:nil, nil] show];

    });

    self.window = [[UIWindow alloc] initWithFrame:
                   [[UIScreen mainScreen] bounds]];
```

```
    self.window.backgroundColor = [UIColor whiteColor];
    [self.window makeKeyAndVisible];
    return YES;
}
```

Поскольку функция dispatch_group_async_f принимает функцию на языке C как блок кода для исполнения, у функции C должна быть ссылка на self, чтобы она могла активизировать методы экземпляра актуального объекта, где реализована функция C. Вот почему self передается как указатель контекста в функции dispatch_group_async_f. Подробнее о контекстах и функциях C рассказано в разделе 7.4.

После того как все поставленные задачи будут завершены, пользователь увидит примерно такую картинку, как на рис. 7.3.

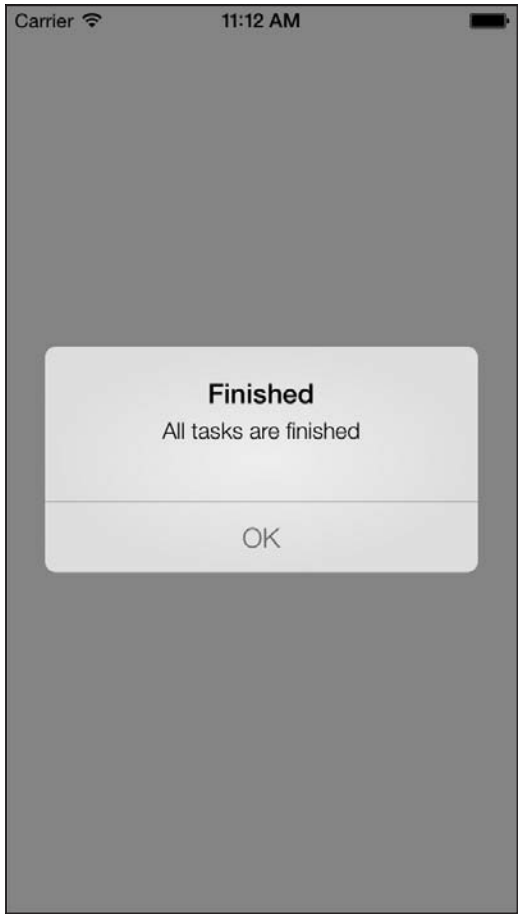


**Рис. 7.3.** Управление группой задач в GCD

## См. также

Раздел 7.4.

# 7.10. Создание собственных диспетчерских очередей с помощью GCD

## Постановка задачи

Требуется создавать собственные диспетчерские очереди с уникальными именами.

## Решение

Воспользуйтесь функцией `dispatch_queue_create`.

## Обсуждение

Работая с GCD, вы можете создавать собственные последовательные диспетчерские очереди (см. раздел 7.0, где подробно рассказано о последовательных очередях). Задачи в последовательных диспетчерских очередях выполняются по принципу «первым пришел — первым обслужен» (FIFO). Но асинхронные задачи, выполняемые в последовательных очередях, не осуществляются в главном потоке, благодаря чему последовательные очереди очень хорошо подходят для решения параллельных FIFO-задач.

Все синхронные задачи, передаваемые в последовательную очередь, будут выполняться в том потоке, который в данный момент используется кодом, подающим задачу в очередь, — всякий раз, когда это возможно. Но асинхронные задачи, подаваемые в последовательную очередь, будут выполняться не в главном, а в каком-то другом потоке.

Для создания последовательных очередей мы будем пользоваться функцией `dispatch_queue_create`. Первый параметр этой функции — строка на языке C (`char *`), которая уникально идентифицирует данную последовательную очередь в *системе*. Я делаю особый акцент на *системе*, потому что данный идентификатор действует в рамках всей системы. Это означает, что если ваше приложение создает новую последовательную очередь с идентификатором `serialQueue1` и то же самое делает какое-то другое приложение, GCD не сможет зафиксировать акт создания такой одноименной последовательной очереди. Поэтому Apple настоятельно рекомендует, чтобы идентификаторы записывались в формате «обратное доменное имя» (Reverse DNS Format). Идентификаторы в формате обратных доменных имен обычно составляются по следующему принципу: `com.COMPANY.PRODUCT.IDENTIFIER`. Например, я могу создать две последовательные очереди и присвоить им следующие имена:

```
com.pixolity.GCD.serialQueue1
com.pixolity.GCD.serialQueue2
```

После того как последовательная очередь будет готова, можно приступать к диспетчеризации задач в эту очередь, пользуясь различными функциями GCD, изученными в этой книге.

Пожалуй, самое время для примера. Вот он!

```
- (BOOL) application:(UIApplication *)application
  didFinishLaunchingWithOptions:(NSDictionary *)launchOptions{
    dispatch_queue_t firstSerialQueue =
    dispatch_queue_create("com.pixolity.GCD.serialQueue1", 0);

    dispatch_async(firstSerialQueue, ^{
      NSUInteger counter = 0;
      for (counter = 0;
           counter < 5;
           counter++){
        NSLog(@"First iteration, counter = %lu", (unsigned long)counter);
      }
    });

    dispatch_async(firstSerialQueue, ^{
      NSUInteger counter = 0;
      for (counter = 0;
           counter < 5;
           counter++){
        NSLog(@"Second iteration, counter = %lu", (unsigned long)counter);
      }
    });

    dispatch_async(firstSerialQueue, ^{
      NSUInteger counter = 0;
      for (counter = 0;
           counter < 5;
           counter++){
        NSLog(@"Third iteration, counter = %lu", (unsigned long)counter);
      }
    });

    self.window = [[UIWindow alloc] initWithFrame:
    [[UIScreen mainScreen] bounds]];
    self.window.backgroundColor = [UIColor whiteColor];
    [self.window makeKeyAndVisible];
    return YES;
}
```

Запустив этот код, обратите внимание на то, какая информация выводится в окне консоли. Результаты будут примерно такими:

```
First iteration, counter = 0
First iteration, counter = 1
First iteration, counter = 2
First iteration, counter = 3
First iteration, counter = 4
```

```
Second iteration, counter = 0
Second iteration, counter = 1
Second iteration, counter = 2
Second iteration, counter = 3
Second iteration, counter = 4
Third iteration, counter = 0
Third iteration, counter = 1
Third iteration, counter = 2
Third iteration, counter = 3
Third iteration, counter = 4
```

Очевидно, что, хотя мы и направляли блоковые объекты в последовательную очередь асинхронно, очередь выполняла их код в порядке «первым пришел — первым обслужен». Мы можем изменить этот пример с кодом так, чтобы пользоваться функцией `dispatch_async_f` вместо `dispatch_async`:

```
void firstIteration(void *paramContext){

  NSUInteger counter = 0;
  for (counter = 0;
       counter < 5;
       counter++){
    NSLog(@"First iteration, counter = %lu", (unsigned long)counter);
  }
}

void secondIteration(void *paramContext){

  NSUInteger counter = 0;
  for (counter = 0;
       counter < 5;
       counter++){
    NSLog(@"Second iteration, counter = %lu", (unsigned long)counter);
  }
}

void thirdIteration(void *paramContext){

  NSUInteger counter = 0;
  for (counter = 0;
       counter < 5;
       counter++){
    NSLog(@"Third iteration, counter = %lu", (unsigned long)counter);
  }
}

- (BOOL)            application:(UIApplication *)application
  didFinishLaunchingWithOptions:(NSDictionary *)launchOptions{

  dispatch_queue_t firstSerialQueue =
  dispatch_queue_create("com.pixolity.GCD.serialQueue1", 0);
```

```
  dispatch_async_f(firstSerialQueue, NULL, firstIteration);
  dispatch_async_f(firstSerialQueue, NULL, secondIteration);
  dispatch_async_f(firstSerialQueue, NULL, thirdIteration);

  self.window = [[UIWindow alloc] initWithFrame:
                 [[UIScreen mainScreen] bounds]];
  self.window.backgroundColor = [UIColor whiteColor];
  [self.window makeKeyAndVisible];
  return YES;
}
```

# 7.11. Синхронное выполнение задач с помощью операций

## Постановка задачи

Необходимо синхронно выполнить серию задач.

## Решение

Создавайте операции и запускайте их вручную:

```
@interface AppDelegate ()
@property (nonatomic, strong) NSInvocationOperation *simpleOperation;
@end
```

Реализация делегата приложения такова:

```
- (void) simpleOperationEntry:(id)paramObject{

  NSLog(@"Parameter Object = %@", paramObject);
  NSLog(@"Main Thread = %@", [NSThread mainThread]);
  NSLog(@"Current Thread = %@", [NSThread currentThread]);

}

- (BOOL)            application:(UIApplication *)application
  didFinishLaunchingWithOptions:(NSDictionary *)launchOptions{

  NSNumber *simpleObject = [NSNumber numberWithInteger:123];

  self.simpleOperation = [[NSInvocationOperation alloc]
                          initWithTarget:self
                          selector:@selector(simpleOperationEntry:)
                          object:simpleObject];

  [self.simpleOperation start];

  self.window = [[UIWindow alloc] initWithFrame:
```

```
                [[UIScreen mainScreen] bounds]];
  self.window.backgroundColor = [UIColor whiteColor];
  [self.window makeKeyAndVisible];
  return YES;
}
```

Вывод этой программы (в окне консоли) будет примерно таким:

```
Parameter Object = 123
Main Thread = <NSThread: 0x6810280>{name = (null), num = 1}
Current Thread = <NSThread: 0x6810280>{name = (null), num = 1}
```

Из имени данного класса (`NSInvocationOperation`) понятно[1], что основное применение объекта такого типа связано с активизацией метода в объекте. Это наиболее непосредственный способ активизации метода в объекте с помощью операций.

## Обсуждение

Операция активизации, как объяснялось в разделе 7.0, позволяет активизировать метод в объекте. «Что же в этом особенного?» — спросите вы. Потенциал активизирующей операции можно продемонстрировать, когда такая операция добавляется в операционную очередь. Примененная вместе с операционной очередью, активизирующая операция может асинхронно запустить метод в заданном объекте параллельно тому потоку, который начал операцию. Внимательно рассмотрев вывод с консоли (приведенный в подразделе «Решение» данного раздела), вы заметите, что актуальный поток в методе, запущенный активизирующей операцией, равен главному потоку. Действительно, главный поток в методе `application:didFinishLaunchingWithOptions:` запускает операцию, пользуясь ее методом `start`. В разделе 7.12 мы научимся эффективно использовать операционные очереди для асинхронного выполнения задач.

Кроме активизирующих операций, вы можете применять блоковые или обычные операции для синхронного выполнения задач. Вот пример использования блоковой операции для подсчета чисел от 0 до 999 (это происходит в `.h`-файле делегата приложения):

```
@interface AppDelegate ()
@property (nonatomic, strong) NSBlockOperation *simpleOperation;
@end

@implementation AppDelegate
```

А вот реализация делегата приложения (`.m`-файл):

```
- (BOOL)            application:(UIApplication *)application
  didFinishLaunchingWithOptions:(NSDictionary *)launchOptions{

  self.simpleOperation = [NSBlockOperation blockOperationWithBlock:^{
    NSLog(@"Main Thread = %@", [NSThread mainThread]);
```

[1] Invocation (англ.) — «активизация». — *Примеч. пер.*

```
    NSLog(@"Current Thread = %@", [NSThread currentThread]);
    NSUInteger counter = 0;
    for (counter = 0;
         counter < 1000;
         counter++){
      NSLog(@"Count = %lu", (unsigned long)counter);
    }
  }];

  /* Запуск операции. */
  [self.simpleOperation start];
  /* Выводим что-нибудь на консоль, просто чтобы проверить,
  должны мы дожидаться, пока выполнится блок кода, или нет.*/
  NSLog(@"Main thread is here");

  self.window = [[UIWindow alloc] initWithFrame:
                 [[UIScreen mainScreen] bounds]];
  self.window.backgroundColor = [UIColor whiteColor];
  [self.window makeKeyAndVisible];
  return YES;
}
```

Если запустить приложение, мы увидим, что на экране выводятся значения от 0 до 999, а за ними следует сообщение Main thread is here (Это главный поток):

```
Main Thread = <NSThread: 0x6810280>{name = (null), num = 1}
Current Thread = <NSThread: 0x6810280>{name = (null), num = 1}
...
Count = 991
Count = 992
Count = 993
Count = 994
Count = 995
Count = 996
Count = 997
Count = 998
Count = 999
Main thread is here
```

Итак, убеждаемся, что, поскольку блоковая операция была запущена в методе `application:didFinishLaunchingWithOptions:`, который сам работает в главном потоке, код внутри блока также выполняется в главном потоке. Основные сведения, которые мы получаем из этих регистрационных записей (логов), сводятся к следующему: операция блокировала главный поток, и потребовалось вернуться к выполнению кода основного потока после того, как была завершена работа блоковой операции. Это образец очень непрофессионального программирования. На самом деле программисты, работающие с iOS, должны идти на любые уловки и пользоваться любыми известными им приемами, чтобы обеспечивать отклик основного потока в любой момент и чтобы этот поток мог заниматься своим основным делом — обработкой пользовательского ввода. Вот что об этом пишет Apple.

«Необходимо внимательно отслеживать, какие задачи вы решаете в главном потоке вашего приложения. Именно в главном потоке ваша программа отрабатывает события касания и другой пользовательский ввод. Чтобы гарантировать, что приложение в любой момент будет откликаться на действия пользователя, никогда не следует загружать главный поток выполнением долговременных задач либо выполнением задач с потенциально неопределенным концом. Таковы, в частности, задачи, связанные с доступом к сети. Напротив, подобные задачи следует решать в фоновых потоках. Оптимальный способ решения таких задач — заключение их в объект операции и добавление этого объекта к операционной очереди. Но вы можете и сами создавать потоки вручную».

Подробнее эта тема рассматривается в документе Performance Tuning («Повышение производительности») в справочной библиотеке iOS. Документ расположен по адресу https://developer.apple.com/library/ios/documentation/iphone/conceptual/iphoneosprogrammingguide/PerformanceTuning/PerformanceTuning.html#//apple_ref/doc/uid/TP40007072-CH8-SW1.

Кроме вызовов и блоковых операций, вы можете также создавать подклассы от `NSOperation` и выполнять вашу задачу в этом классе. Перед тем как переходить к работе, обратите внимание на некоторые нюансы, связанные с созданием подклассов от `NSOperation`.

Если вы не планируете пользоваться операционной очередью, то необходимо открепить от вашего потока новый поток. Это делается в методе `start`, относящемся к операции. Если вы не хотите пользоваться операционной очередью, а также не собираетесь выполнять данную операцию асинхронно с другими операциями, запускаемыми вручную, то можно просто вызвать метод `main` операции в методе `start`.

В реализации операции необходимо переопределить два важных метода экземпляра `NSOperation` — методы `isExecuting` и `isFinished`. Их может вызывать любой другой объект. В этих методах необходимо возвращать потоковобезопасное значение, которым можно будет управлять прямо из операции. Как только операция начинается, она должна посредством механизма «уведомления наблюдателей об изменениях в свойствах наблюдаемого объекта» (KVO) сообщать всем слушателям о том, что вы изменяете возвращаемые значения для двух этих методов. В примере с кодом мы рассмотрим, как это происходит на практике.

В методе `main`, относящемся к операции, необходимо создать собственный автоматически высвобождаемый пул на случай, если когда-нибудь в будущем операция будет добавлена в операционную очередь. Необходимо убедиться в том, что операции можно задействовать обоими способами — как при запуске вручную, так и при запуске в рамках операционной очереди.

У вас должен быть метод-инициализатор для ваших операций. Это обязательно должен быть специальный метод, выделенный под конкретную операцию. Все остальные методы-инициализаторы, в том числе применяемый по умолчанию метод `init`, должны вызывать вышеупомянутый специальный инициализатор, который содержит наибольшее количество параметров. Другие методы-инициализаторы должны гарантировать, что они передают подходящие параметры методу-инициализатору (если вообще передают).

Вот объявление объекта операции (.h-файл):

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

@interface CountingOperation : NSOperation

/* Выделенный инициализатор */
- (id) initWithStartingCount:(NSUInteger)paramStartingCount
                 endingCount:(NSUInteger)paramEndingCount;

@end
```

Реализация операции (записываемая в .m-файле) несколько длинновата, но, надеюсь, вполне понятна:

```
#import "CountingOperation.h"

@implementation CountingOperation

@property (nonatomic, unsafe_unretained) NSUInteger startingCount;
@property (nonatomic, unsafe_unretained) NSUInteger endingCount;
@property (nonatomic, unsafe_unretained, getter=isFinished) BOOL finished;
@property (nonatomic, unsafe_unretained, getter=isExecuting) BOOL executing;
@end

@implementation CountingOperation

- (instancetype) init {
  return([self initWithStartingCount:0
  endingCount:1000]);
}

- (instancetype) initWithStartingCount:(NSUInteger)paramStartingCount
  endingCount:(NSUInteger)paramEndingCount{
    self = [super init];

    if (self != nil){

      /* Сохраните эти значения для главного метода. */
      startingCount = paramStartingCount;
      endingCount = paramEndingCount;

    }

    return(self);

}

- (void) main {

  @try {
```

```
    /* Это автоматически высвобождаемый пул. */
    @autoreleasepool {
      /* Сохраняем здесь локальную переменную, которая
         должна быть установлена в YES всякий раз, когда
         мы завершаем выполнение задачи. */
      BOOL taskIsFinished = NO;

      /* Создаем здесь цикл while, существующий лишь в том случае,
         когда переменная taskIsFinished устанавливается в YES
         или операция отменяется. */
      while (taskIsFinished == NO &&
             [self isCancelled] == NO){

      /* Здесь выполняется задача. */
        NSLog(@"Main Thread = %@", [NSThread mainThread]);
        NSLog(@"Current Thread = %@", [NSThread currentThread]);
        NSUInteger counter = startingCount;
        for (counter = startingCount;
             counter < endingCount;
             counter++){
          NSLog(@"Count = %lu", (unsigned long)counter);
        }
        /* Очень важно. Здесь мы можем выйти из цикла, по-прежнему
           соблюдая правила, по которым отменяются операции. */
        taskIsFinished = YES;

        }

        /* Соответствие KVO. Генерируем требуемые уведомления KVO. */
        [self willChangeValueForKey:@"isFinished"];
        [self willChangeValueForKey:@"isExecuting"];
        finished = YES;
        executing = NO;
        [self didChangeValueForKey:@"isFinished"];
        [self didChangeValueForKey:@"isExecuting"];
    }
  }
  @catch (NSException * e) {
    NSLog(@"Exception %@", e);
  }

}

@end
```

Операцию можно начать так:

```
@interface AppDelegate ()
@property (nonatomic, strong) CountingOperation *simpleOperation;
@end

@implementation AppDelegate
```

```
- (BOOL)            application:(UIApplication *)application
  didFinishLaunchingWithOptions:(NSDictionary *)launchOptions{

  self.simpleOperation = [[CountingOperation alloc] initWithStartingCount:0
                                                              endingCount:1000];

  [self.simpleOperation start];

  NSLog(@"Main thread is here");

  self.window = [[UIWindow alloc] initWithFrame:
                 [[UIScreen mainScreen] bounds]];
  self.window.backgroundColor = [UIColor whiteColor];
  [self.window makeKeyAndVisible];
  return YES;
}

@end
```

Запустив данный код, мы увидим в окне консоли следующие результаты, точно как при применении блоковой операции:

```
Main Thread = <NSThread: 0x6810260>{name = (null), num = 1}
Current Thread = <NSThread: 0x6810260>{name = (null), num = 1}
...
Count = 993
Count = 994
Count = 995
Count = 996
Count = 997
Count = 998
Count = 999
Main thread is here
```

## См. также

Раздел 7.12.

# 7.12. Асинхронное выполнение задач с помощью операций

## Постановка задачи

Требуется параллельно выполнять операции.

## Решение

Воспользуйтесь операционными очередями. В качестве альтернативы можно создавать подклассы от `NSOperation` и откреплять новый поток в методе `main`.

## Обсуждение

Как говорилось в разделе 7.11, операции по умолчанию работают в том потоке, который вызывает метод `start`. Обычно операции запускаются в основном потоке, но в то же время мы ожидаем, что операции будут выполняться в собственных потоках и, соответственно, не будут тратить процессорное время, уделяемое главному потоку. Наилучшим решением для обеспечения такой работы будет применение операционных очередей. Однако если вы хотите управлять своими операциями вручную, чего бы я не рекомендовал, то можно было бы создавать подклассы от `NSOperation` и откреплять новый поток в главном методе. Подробнее об открепленных потоках поговорим в разделе 7.15.

Идем дальше. Попробуем воспользоваться операционной очередью и добавим к ней две простые инициирующие операции (подробнее об инициирующих операциях рассказано в разделе 7.0). Дополнительные примеры кода, описывающие инициирующие операции, имеются в разделе 7.11. Вот объявление (`.hm`-файл) делегата приложения, в котором используются операционная очередь и две инициирующие операции:

```
@interface AppDelegate ()
@property (nonatomic, strong) NSOperationQueue *operationQueue;
@property (nonatomic, strong) NSInvocationOperation *firstOperation;
@property (nonatomic, strong) NSInvocationOperation *secondOperation;
@end

@implementation AppDelegate
```

А вот и внутренняя часть файла реализации делегата приложения:

```
- (void) firstOperationEntry:(id)paramObject{

  NSLog(@"%s", __FUNCTION__);
  NSLog(@"Parameter Object = %@", paramObject);
  NSLog(@"Main Thread = %@", [NSThread mainThread]);
  NSLog(@"Current Thread = %@", [NSThread currentThread]);

}

- (void) secondOperationEntry:(id)paramObject{

  NSLog(@"%s", __FUNCTION__);
  NSLog(@"Parameter Object = %@", paramObject);
  NSLog(@"Main Thread = %@", [NSThread mainThread]);
  NSLog(@"Current Thread = %@", [NSThread currentThread]);

}

- (BOOL)            application:(UIApplication *)application
  didFinishLaunchingWithOptions:(NSDictionary *)launchOptions{

NSNumber *firstNumber = @111;
```

```
NSNumber *secondNumber = @222;

  self.firstOperation =[[NSInvocationOperation alloc]
                        initWithTarget:self
                        selector:@selector(firstOperationEntry:)
                        object:firstNumber];

  self.secondOperation = [[NSInvocationOperation alloc]
                          initWithTarget:self
                          selector:@selector(secondOperationEntry:)
                          object:secondNumber];

  self.operationQueue = [[NSOperationQueue alloc] init];

  /* Добавляем операции в очередь. */
  [self.operationQueue addOperation:self.firstOperation];
  [self.operationQueue addOperation:self.secondOperation];

  NSLog(@"Main thread is here");

  self.window = [[UIWindow alloc] initWithFrame:
                 [[UIScreen mainScreen] bounds]];
  self.window.backgroundColor = [UIColor whiteColor];
  [self.window makeKeyAndVisible];
  return YES;
}
```

Вот что происходит в реализации данного кода.

У нас есть два метода, `firstOperationEntry:` и `secondOperationEntry:`. Каждый из этих методов принимает в качестве параметра объект и выводит в окне консоли информацию об актуальном потоке, главном потоке и этом параметре. Это входные методы инициирующих операций, которые будут добавляться в операционную очередь.

Мы инициализируем два метода типа `NSInvocationOperation` и задаем целевой селектор в точке входа каждой операции (эти точки входа были описаны выше).

Затем инициализируем объект типа `NSOperationQueue`. (Он может создаваться и до того, как созданы методы входа.) Объект очереди будет обеспечивать параллелизм в работе операционных объектов. На данном этапе операционная очередь может немедленно начать (а может и не начать) запускать инициирующие операции, пользуясь их методами `start`. При этом очень важно помнить, что после добавления операции в операционную очередь от вас не требуется запускать операции вручную. Обеспечением запуска занимается операционная очередь.

Итак, еще раз запустим код примера и посмотрим, что же у нас на консоли:

```
[Running_Tasks_Asynchronously_with_OperationsAppDelegate firstOperationEntry:]
Main thread is here
Parameter Object = 111
[Running_Tasks_Asynchronously_with_OperationsAppDelegate secondOperationEntry:]
Main Thread = <NSThread: 0x6810260>{name = (null), num = 1}
```

```
Parameter Object = 222
Current Thread = <NSThread: 0x6805c20>{name = (null), num = 3}
Main Thread = <NSThread: 0x6810260>{name = (null), num = 1}
Current Thread = <NSThread: 0x6b2d1d0>{name = (null), num = 4}
```

Блестяще! Это доказывает, что инициирующие операции параллельно выполняются каждая в своем потоке и в то же время параллельно главному потоку, вообще не блокируя его. Теперь еще пару раз прогоним этот же код и посмотрим, какой вывод будет появляться в окне консоли. В таком случае вы можете получить совершенно иной результат, например:

```
Main thread is here
[Running_Tasks_Asynchronously_with_OperationsAppDelegate firstOperationEntry:]
[Running_Tasks_Asynchronously_with_OperationsAppDelegate secondOperationEntry:]
Parameter Object = 111
Main Thread = <NSThread: 0x6810260>{name = (null), num = 1}
Current Thread = <NSThread: 0x68247c0>{name = (null), num = 3}
Parameter Object = 222
Main Thread = <NSThread: 0x6810260>{name = (null), num = 1}
Current Thread = <NSThread: 0x6819b00>{name = (null), num = 4}
```

Очевидно, что главный поток не блокируется и что обе инициирующие операции работают параллельно с главным потоком. Это доказывает, что в операционной очереди сохраняется параллелизм даже тогда, когда в нее добавляются две непараллельные операции. Операционная очередь управляет потоками, необходимыми для осуществления операций.

Если бы вы создавали подклассы от `NSOperation` и добавляли в операционную очередь экземпляры нового класса, то ситуация складывалась бы несколько иначе. Не забывайте о некоторых моментах.

Если обычные операции, являющиеся подклассами от `NSOperation`, добавлять в операционную очередь, то они будут работать асинхронно. Поэтому необходимо переопределить метод экземпляра `isConcurrent`, относящийся к классу `NSOperation`, и возвратить значение `YES`.

Необходимо подготовить операцию к отмене, периодически проверяя значение метода `isCancelled` при осуществлении основной задачи операции, а также в методе `start` еще до запуска самой операции. В таком случае метод `start` вызывается операционной очередью после того, как операция будет добавлена в очередь. В этом методе проверяется, не отменена ли операция. Это делается с помощью метода `isCancelled`. Если операция отменена, просто верните такое значение от метода `start`. В противном случае вызовите метод `main` из метода `start`.

Переопределите метод `main` собственной реализацией основной задачи, которую должна выполнять операция. Обязательно выделите и инициализируйте в этом методе ваш собственный автоматически высвобождаемый пул и высвободите его непосредственно перед актом возврата.

Переопределите методы `isFinished` и `isExecuting` операции и верните соответствующие логические (`BOOL`) значения, показывающие, завершена операция или продолжается в настоящий момент.

Вот объявление операции (.h-файл):

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

@interface SimpleOperation : NSOperation

/* Выделенный инициализатор */
- (id) initWithObject:(NSObject *)paramObject;

@end
```

Реализация операции такова:

```
#import "SimpleOperation.h"

@implementation SimpleOperation

- (instancetype) init {
  return([self initWithObject:@123]);
}

- (instancetype) initWithObject:(NSObject *)paramObject{
  self = [super init];
  if (self != nil){
    /* Сохраните эти значения для главного метода. */
    _givenObject = paramObject;
  }
  return(self);
}
- (void) main {

  @try {
    @autoreleasepool {
      /* Сохраняем здесь локальную переменную, которая должна быть
         установлена в YES всякий раз, когда мы завершаем
         выполнение задачи. */
      BOOL taskIsFinished = NO;

      /* Создаем здесь цикл while, существующий лишь в том случае,
         когда переменная taskIsFinished устанавливается в YES
         или операция отменяется. */
      while (taskIsFinished == NO &&
            [self isCancelled] == NO){

        /* Здесь выполняется задача. */
        NSLog(@"%s", __FUNCTION__);
        NSLog(@"Parameter Object = %@", givenObject);
        NSLog(@"Main Thread = %@", [NSThread mainThread]);
        NSLog(@"Current Thread = %@", [NSThread currentThread]);

        /* Очень важно. Здесь мы можем выйти из цикла, по-прежнему
           соблюдая правила, по которым отменяются операции. */
```

```
        taskIsFinished = YES;

}

      /* Соответствие KVO. Генерируем требуемые уведомления KVO. */
      [self willChangeValueForKey:@"isFinished"];
      [self willChangeValueForKey:@"isExecuting"];
      finished = YES;
      executing = NO;
      [self didChangeValueForKey:@"isFinished"];
      [self didChangeValueForKey:@"isExecuting"];
    }
  }
  @catch (NSException * e) {
    NSLog(@"Exception %@", e);
  }
}

- (BOOL)  isConcurrent{
  return YES;
}

@end
```

Теперь этот операционный класс можно использовать в любом другом классе, например в делегате вашего приложения. Вот объявление делегата приложения, в котором задействуется этот новый класс операции, добавляемый в операционную очередь:

```
@interface AppDelegate ()
@property (nonatomic, strong) NSOperationQueue *operationQueue;
@property (nonatomic, strong) SimpleOperation *firstOperation;
@property (nonatomic, strong) SimpleOperation *secondOperation;
@end

@implementation AppDelegate
```

Реализация делегата приложения такова:

```
- (BOOL)            application:(UIApplication *)application
  didFinishLaunchingWithOptions:(NSDictionary *)launchOptions{

  NSNumber *firstNumber = @111;
  NSNumber *secondNumber = @222;

  self.firstOperation = [[SimpleOperation alloc]
                         initWithObject:firstNumber];
  self.secondOperation = [[SimpleOperation alloc]
                          initWithObject:secondNumber];

  self.operationQueue = [[NSOperationQueue alloc] init];
```

```
  /* Добавляем операции в очередь. */
  [self.operationQueue addOperation:self.firstOperation];
  [self.operationQueue addOperation:self.secondOperation];

  NSLog(@"Main thread is here");

  self.window = [[UIWindow alloc] initWithFrame:
                 [[UIScreen mainScreen] bounds]];
  self.window.backgroundColor = [UIColor whiteColor];
  [self.window makeKeyAndVisible];
  return YES;
}
```

В окне консоли отобразятся результаты, подобные уже виденным ранее — тем, которые мы получали при применении параллельных инициирующих операций:

```
Main thread is here
-[SimpleOperation main]
-[SimpleOperation main]
Parameter Object = 222
Parameter Object = 222
Main Thread = <NSThread: 0x6810260>{name = (null), num = 1}
Main Thread = <NSThread: 0x6810260>{name = (null), num = 1}
Current Thread = <NSThread: 0x6a10b90>{name = (null), num = 3}
Current Thread = <NSThread: 0x6a13f50>{name = (null), num = 4}
```

## См. также

Разделы 7.11 и 7.15.

# 7.13. Создание зависимости между операциями

## Постановка задачи

Необходимо начать выполнение определенной задачи только после того, как завершится выполнение другой определенной задачи.

## Решение

Если операция B может начать выполнение содержащейся в ней задачи только после того, как операция A выполнит свою задачу, то операция B должна добавить к себе операцию A в качестве зависимой. Это делается с помощью метода экземпляра `addDependency:`, относящегося к классу `NSOperation`:

```
[self.firstOperation addDependency:self.secondOperation];
```

Свойства `firstOperation` и `secondOperation` относятся к типу `NSInvocationOperation`, подробнее об этом мы поговорим в подразделе «Обсуждение» данного раздела. В приведенном примере кода первая операция, находящаяся в операционной очереди, не будет выполняться до тех пор, пока не будет выполнена задача второй операции.

## Обсуждение

Выполнение операции не начинается до тех пор, пока не будут успешно завершены все операции, от которых она зависит. По умолчанию после инициализации операция не связана зависимостями с какими-либо другими операциями.

Если мы хотим внедрить зависимости в пример с кодом, описанный в разделе 7.12, то можем немного изменить реализацию делегата приложения и воспользоваться методом экземпляра `addDependency:`, чтобы первая операция дождалась окончания выполнения второй операции:

```
#import "AppDelegate.h"

@interface AppDelegate ()
@property (nonatomic, strong) NSInvocationOperation *firstOperation;
@property (nonatomic, strong) NSInvocationOperation *secondOperation;
@property (nonatomic, strong) NSOperationQueue *operationQueue;
@end

@implementation AppDelegate
- (void) firstOperationEntry:(id)paramObject{

  NSLog(@"First Operation - Parameter Object = %@",
        paramObject);

  NSLog(@"First Operation - Main Thread = %@",
        [NSThread mainThread]);

  NSLog(@"First Operation - Current Thread = %@",
        [NSThread currentThread]);

}

- (void) secondOperationEntry:(id)paramObject{

  NSLog(@"Second Operation - Parameter Object = %@",
        paramObject);

  NSLog(@"Second Operation - Main Thread = %@",
        [NSThread mainThread]);

  NSLog(@"Second Operation - Current Thread = %@",
        [NSThread currentThread]);

}
```

```
- (BOOL)            application:(UIApplication *)application
  didFinishLaunchingWithOptions:(NSDictionary *)launchOptions{

  NSNumber *firstNumber = @111;
  NSNumber *secondNumber = @222;

  self.firstOperation = [[NSInvocationOperation alloc]
                         initWithTarget:self
                         selector:@selector(firstOperationEntry:)
                         object:firstNumber];

  self.secondOperation = [[NSInvocationOperation alloc]
                          initWithTarget:self
                          selector:@selector(secondOperationEntry:)
                          object:secondNumber];

  [self.firstOperation addDependency:self.secondOperation];

  self.operationQueue = [[NSOperationQueue alloc] init];

  /* Добавляем операции в очередь. */
  [self.operationQueue addOperation:self.firstOperation];
  [self.operationQueue addOperation:self.secondOperation];

  NSLog(@"Main thread is here");

  self.window = [[UIWindow alloc] initWithFrame:
                 [[UIScreen mainScreen] bounds]];
  self.window.backgroundColor = [UIColor whiteColor];
  [self.window makeKeyAndVisible];
  return YES;
}
```

Теперь после запуска программ вы увидите в окне консоли примерно следующее:

```
Second Operation - Parameter Object = 222
Main thread is here
Second Operation - Main Thread = <NSThread: 0x6810250>{name = (null),
num = 1}
Second Operation - Current Thread = <NSThread: 0x6836ab0>{name = (null),
num = 3}
First Operation - Parameter Object = 111
First Operation - Main Thread = <NSThread: 0x6810250>{name = (null),
num = 1}
First Operation - Current Thread = <NSThread: 0x6836ab0>{name = (null),
num = 3}
```

Вполне очевидно, что, хотя операционная очередь пытается параллельно вести обе операции, первая операция находится в зависимости от второй, следовательно,

вторая операция должна завершиться, и только после этого может начаться первая операция.

Если вы в любой момент пожелаете разорвать зависимость между двумя операциями, воспользуйтесь методом экземпляра `removeDependency:`, относящимся к объекту операции.

## См. также

Раздел 7.12.

# 7.14. Создание таймеров

## Постановка задачи

Требуется многократно выполнять определенную задачу после заданной задержки. Например, вы хотите обновлять вид на экране устройства каждую секунду, пока работает ваше приложение.

## Решение

Воспользуйтесь таймером:

```
- (void) paint:(NSTimer *)paramTimer{
  /* Делаем здесь что-либо. */
  NSLog(@"Painting");
}

- (void) startPainting{

  self.paintingTimer = [NSTimer
                        scheduledTimerWithTimeInterval:1.0
                        target:self
                        selector:@selector(paint:)
                        userInfo:nil
                        repeats:YES];

}

- (void) stopPainting{
  if (self.paintingTimer != nil){
    [self.paintingTimer invalidate];
  }
}

- (void)applicationWillResignActive:(UIApplication *)application{
  [self stopPainting];
```

```
}

- (void)applicationDidBecomeActive:(UIApplication *)application{
  [self startPainting];
}
```

Кроме того, метод `invalidate` будет высвобождать таймер сам и нам не придется делать это вручную. Как видите, мы определили свойство `paintingTimer`, которое следующим образом определяется в заголовочном файле (`.h`-файле):

```
#import "AppDelegate.h"

@interface AppDelegate ()
@property (nonatomic, strong) NSTimer *paintingTimer;
@end

@implementation AppDelegate
```

## Обсуждение

Таймер — это объект, инициирующий определенное событие через заданные временные интервалы. Таймер должен быть запланирован в рабочем цикле. При определении объекта `NSTimer` создается незапланированный таймер, который ничего не делает, но остается в распоряжении программы на случай, если этот таймер понадобится запланировать. Как только будет сделан вызов вида `scheduledTimerWithTimeInterval:target:selector:userInfo:repeats:`, начинается работа запланированного таймера и будет инициировано затребованное вами событие. Запланированным называется такой таймер, который добавлен к рабочему циклу. Чтобы получить любой таймер и инициировать связанное с ним событие, таймер нужно запланировать в рабочем цикле. Это будет продемонстрировано в следующем примере, где мы создадим незапланированный таймер, а затем вручную запланируем его в главном рабочем цикле приложения.

После того как таймер запланирован и добавлен к рабочему циклу — явно или неявно, — он начинает вызывать метод в своем целевом объекте (указываемом программистом) каждые *n* секунд (*n* также указывает программист). Поскольку *n* — это число с плавающей точкой, в данном параметре можно задать долю секунды.

Существуют различные способы создания, инициализации и планирования таймеров. Один из наиболее простых способов связан с использованием метода класса `scheduledTimerWithTimeInterval:target:selector:userInfo:repeats:`, относящегося к классу `NSTimer`. Далее перечислены параметры данного метода:

- `scheduledTimerWithTimeInterval` — количество секунд, в течение которого таймер должен ожидать, прежде чем запустит то или иное событие. Например, если вы хотите, чтобы таймер вызывал метод в своем целевом объекте дважды в секунду, то для этого параметра нужно установить значение `0.5` (1 секунда, деленная на 2). Если вы желаете, чтобы целевой метод вызывался четыре раза в секунду, то этот параметр должен иметь значение `0.25` (1 секунда, деленная на 4);

- target — объект, который будет получать событие;
- selector — сигнатура метода в том целевом объекте, который будет получать событие;
- userInfo — объект, который будет содержаться в таймере для дальнейшего пользования (в целевом методе целевого объекта);
- repeats — параметр указывает, как таймер должен вызывать целевой метод многократно (в таком случае данный параметр получает значение YES) или однократно (тогда этот параметр получит значение NO).

Как только таймер создан и добавлен к рабочему циклу, можно остановиться и высвободить этот таймер, воспользовавшись методом экземпляра invalidate, относящимся к классу NSTimer. Таким образом будет высвобожден не только таймер, но и объект (если имеется объект, который передан таймеру и который требуется сохранять на протяжении всего жизненного цикла таймера; например, объект может быть сообщен параметру userInfo метода класса scheduledTimerWithTimeInterval:target:selector:userInfo:repeats:, относящемуся к классу NSTimer). Если передать параметру repeats значение NO, то таймер самоуничтожится после первого прохода цикла и высвободит любой удерживаемый объект (при его наличии).

Есть и другие методы, с помощью которых можно создать запланированный таймер. Один из них — метод класса scheduledTimerWithTimeInterval:invocation:repeats:, относящийся к классу NSTimer:

```
- (void) paint:(NSTimer *)paramTimer{
  /* Делаем здесь что-либо. */
  NSLog(@"Painting");
}

- (void) startPainting{

  /* Здесь находится селектор, который мы хотим вызвать. */
  SEL selectorToCall = @selector(paint:);

  /* Здесь на основе селектора составляется сигнатура метода.
     Нам известно, что селектор относится к текущему классу,
     поэтому составить сигнатуру метода совсем не сложно. */
  NSMethodSignature *methodSignature =
    [[self class] instanceMethodSignatureForSelector:selectorToCall];

  /* Теперь основываем активизацию на сигнатуре метода. Данная активизация
     требуется нам для того, чтобы запланировать таймер. */
  NSInvocation *invocation =
    [NSInvocation invocationWithMethodSignature:methodSignature];
  [invocation setTarget:self];
  [invocation setSelector:selectorToCall];

  /* Теперь запускаем запланированный таймер. */
  self.paintingTimer = [NSTimer scheduledTimerWithTimeInterval:1.0
                                                    invocation:invocation
```

```
                                                    repeats:YES];

}

- (void) stopPainting{
  if (self.paintingTimer != nil){
    [self.paintingTimer invalidate];
  }
}

- (void)applicationWillResignActive:(UIApplication *)application{
  [self stopPainting];
}

- (void)applicationDidBecomeActive:(UIApplication *)application{
  [self startPainting];
}
```

Планирование таймера можно сравнить с запуском автомобильного двигателя. Запланированный таймер — это работающий мотор. Незапланированный таймер — это мотор, который уже готов завестись, но пока не работает. Мы можем планировать и отменять (распланировать) таймер в любой момент работы приложения, точно так же как можем заводить и глушить двигатель, не выходя из машины. Если вы хотите вручную запланировать таймер на определенный момент жизненного цикла приложения, можно воспользоваться методом класса `timerWithTimeInterval:target:selector:userInfo:repeats:`, относящимся к классу `NSTimer`. Когда придет нужный момент, можно добавить таймер к интересующему вас рабочему циклу:

```
- (void) startPainting{

  self.paintingTimer = [NSTimer timerWithTimeInterval:1.0
                                               target:self
                                             selector:@selector(paint:)
                                             userInfo:nil
                                              repeats:YES];

  /* Здесь выполняется обработка, и когда наступает нужный момент,
     задействуется метод экземпляра addTimer:forMode, относящийся к классу
     NSRunLoop, чтобы запланировать данный таймер в этом рабочем цикле. */

  [[NSRunLoop currentRunLoop] addTimer:self.paintingTimer
                               forMode:NSDefaultRunLoopMode];

}
```

Методы класса currentRunLoop и mainRunLoop, относящиеся к классу NSRunLoop, возвращают соответственно актуальный и главный рабочие циклы конкретного приложения, что понятно из их названий[1].

[1] Main (англ.) — «главный», run (англ.) — «рабочий», loop (англ.) — «цикл». — *Примеч. пер.*

Можно создавать запланированные таймеры с применением активизации, воспользовавшись вариантом с методом scheduledTimerWithTimeInterval:invocation:repeats:. С тем же успехом можно пользоваться методом класса timerWithTimeInterval:invocation:repeats:, относящимся к классу NSTimer, чтобы создать незапланированный таймер — также с применением активизации:

```
- (void) paint:(NSTimer *)paramTimer{
  /* Делаем здесь что-нибудь. */
  NSLog(@"Painting");
}

- (void) startPainting{

  /* Здесь находится селектор, который мы хотим вызвать. */
  SEL selectorToCall = @selector(paint:);

  /* Здесь на основе селектора составляется сигнатура метода.
     Нам известно, что селектор относится к текущему классу,
     поэтому составить сигнатуру метода совсем не сложно. */
  NSMethodSignature *methodSignature =
  [[self class] instanceMethodSignatureForSelector:selectorToCall];

  /* Теперь основываем активизацию на сигнатуре метода. Данная активизация
     требуется нам для того, чтобы запланировать таймер. */
  NSInvocation *invocation =
  [NSInvocation invocationWithMethodSignature:methodSignature];

  [invocation setTarget:self];
  [invocation setSelector:selectorToCall];

  self.paintingTimer = [NSTimer timerWithTimeInterval:1.0
                                           invocation:invocation
                                              repeats:YES];;

  /* Здесь выполняется обработка, и когда наступает нужный момент,
     задействуется метод экземпляра addTimer:forMode, относящийся к классу
     NSRunLoop, чтобы запланировать данный таймер в данном рабочем цикле. */

  [[NSRunLoop currentRunLoop] addTimer:self.paintingTimer
                               forMode:NSDefaultRunLoopMode];

}

- (void) stopPainting{
  if (self.paintingTimer != nil){
    [self.paintingTimer invalidate];
  }
}

- (void)applicationWillResignActive:(UIApplication *)application{
  [self stopPainting];
```

```
}

- (void)applicationDidBecomeActive:(UIApplication *)application{
  [self startPainting];
}
```

Целевой метод таймера получает экземпляр таймера, вызывающий его в качестве параметра. Например, метод `paint:`, показанный в начале данного раздела, демонстрирует, как таймер передается своему целевому методу — по умолчанию он (таймер) выступает в качестве единственного параметра целевого метода:

```
- (void) paint:(NSTimer *)paramTimer{
  /* Что-то здесь делаем. */
  NSLog(@"Painting");
}
```

Данный параметр дает нам ссылку на таймер, запускающий этот метод. Вы можете, например, при необходимости не допустить повторного запуска таймера — для этого используется метод `invalidate`. Кроме того, можно активизировать метод `userInfo` экземпляра класса `NSTimer`, чтобы получить объект, удерживаемый таймером (если такой объект имеется). Здесь мы имеем дело с обычным объектом, передаваемым методам инициализации `NSTimer`, затем этот объект передается непосредственно таймеру для дальнейшего пользования.

# 7.15. Параллельное программирование с использованием потоков

## Постановка задачи

Необходимо обеспечить максимально полный контроль над отдельными задачами, выполняемыми в приложении. Например, вам может быть необходимо выполнить объемные расчеты, затребованные пользователем, но в то же время нужно освободить главный поток — поток пользовательского интерфейса — для взаимодействия с пользователем и решения других задач.

## Решение

В приложении воспользуйтесь потоками. Это делается примерно так:

```
- (void) downloadNewFile:(id)paramObject{

  @autoreleasepool {
    NSString *fileURL = (NSString *)paramObject;

    NSURL    *url = [NSURL URLWithString:fileURL];

    NSURLRequest *request = [NSURLRequest requestWithURL:url];

    NSURLResponse *response = nil;
```

```
    NSError       *error = nil;

    NSData *downloadedData =
    [NSURLConnection sendSynchronousRequest:request
                          returningResponse:&response
                                      error:&error];

    if ([downloadedData length] > 0){
      /* Загрузка завершена. */
    } else {
      /* Ничего загружено не было. Проверьте значение Error. */
    }
  }

}

- (void)viewDidLoad {
  [super viewDidLoad];

  NSString *fileToDownload = @"http://www.OReilly.com";

  [NSThread detachNewThreadSelector:@selector(downloadNewFile:)
                           toTarget:self
                         withObject:fileToDownload];

}
```

## Обсуждение

Любое приложение iOS состоит из одного или нескольких потоков. В операционной системе iOS 5 у обычного приложения с одним контроллером вида изначально может быть от одного до пяти потоков, создаваемых системными библиотеками, с которыми связано приложение. Для вашего приложения будет создаваться как минимум один поток независимо от того, собираетесь вы пользоваться несколькими потоками или нет. Этот поток называется основным потоком пользовательского интерфейса и прикрепляется к главному рабочему циклу.

Чтобы оценить, насколько полезны потоки, проведем эксперимент. Предположим, у нас есть три цикла:

```
- (void) firstCounter{

  NSUInteger counter = 0;
  for (counter = 0;
       counter < 1000;
       counter++){
    NSLog(@"First Counter = %lu", (unsigned long)counter);
  }

}
```

```
- (void) secondCounter{

  NSUInteger counter = 0;
  for (counter = 0;
       counter < 1000;
       counter++){
    NSLog(@"Second Counter = %lu", (unsigned long)counter);
  }

}
- (void) thirdCounter{

  NSUInteger counter = 0;
  for (counter = 0;
       counter < 1000;
       counter++){
    NSLog(@"Third Counter = %lu", (unsigned long)counter);
  }

}
```

Очень просто, правда? Все циклы проходят от 0 до 1000, выводя на консоль номера счетчиков. Теперь предположим, что вы хотите, как обычно, запустить эти счетчики:

```
- (void) viewDidLoad{
  [super viewDidLoad];
  [self firstCounter];
  [self secondCounter];
  [self thirdCounter];
}
```

Этот код не обязательно должен находиться в методе viewDidLoad контроллера вида.

Теперь откройте окно консоли и запустите это приложение. Вы увидите, как сначала целиком выполнится первый счетчик, потом второй и, наконец, третий. Это означает, что данные циклы выполняются в одном и том же потоке. Каждый счетчик блокирует исполнение остального кода, относящегося к потоку, пока этот счетчик не завершит свой цикл.

А что, если бы мы захотели запустить все счетчики одновременно? Разумеется, для каждого из них нам пришлось бы создать отдельный поток. Но подождите! Мы ведь уже знаем, что прямо при загрузке приложение само создает для нас потоки. Кроме того, весь код, который мы уже успели создать для приложения, когда бы он ни был написан, исполняется в полученном потоке. Итак, мы уже создали по потоку для первого и второго счетчиков, а третий счетчик будет работать в главном потоке:

```
- (void) firstCounter{

  @autoreleasepool {
```

```
    NSUInteger counter = 0;
    for (counter = 0;
         counter < 1000;
         counter++){
      NSLog(@"First Counter = %lu", (unsigned long)counter);
    }
  }

}

- (void) secondCounter{

  @autoreleasepool {
    NSUInteger counter = 0;
    for (counter = 0;
         counter < 1000;
         counter++){
      NSLog(@"Second Counter = %lu", (unsigned long)counter);
    }
  }

}

- (void) thirdCounter{

  NSUInteger counter = 0;
  for (counter = 0;
       counter < 1000;
       counter++){
    NSLog(@"Third Counter = %lu", (unsigned long)counter);
  }
}

- (void)viewDidLoad {

  [super viewDidLoad];

  [NSThread detachNewThreadSelector:@selector(firstCounter)
                           toTarget:self
                         withObject:nil];

  [NSThread detachNewThreadSelector:@selector(secondCounter)
                           toTarget:self
                         withObject:nil];

  /* Этот код запускаем в главном потоке. */
  [self thirdCounter];

}
```

У метода thirdCounter нет автоматически высвобождаемого пула, поскольку он не работает в новом открепленном потоке. Этот метод будет выполняться в главном потоке приложения, а главный поток располагает автоматически высвобождаемым пулом. Данный пул создается автоматически при написании любого приложения Cocoa Touch.

Ближе к концу кода мы видим вызовы селектора `detachNewThreadSelector`, предназначенные для запуска первого и второго счетчиков в отдельных потоках. Теперь, запустив приложение, вы увидите в окне консоли примерно следующий вывод:

```
Second Counter = 921
Third Counter = 301
Second Counter = 922
Second Counter = 923
Second Counter = 924
First Counter = 956
Second Counter = 925
First Counter = 957
Second Counter = 926
First Counter = 958
Third Counter = 302
Second Counter = 927
Third Counter = 303
Second Counter = 928
```

Иными словами, все три счетчика работают одновременно и их вывод перемежается случайным образом.

Каждый поток должен создавать автоматически высвобождаемый пул. Внутри такого пула содержатся ссылки на объекты, автоматически высвобождаемые до того, как будет высвобожден весь пул. Это очень важный механизм, действующий при управлении памятью с подсчетом ссылок в таких окружениях, как Cocoa Touch, то есть в средах, где объекты могут автоматически высвобождаться. Всякий раз при выделении экземпляра объекта количество ссылок на объект становится равным 1. Если пометить объекты как автоматически высвобождаемые, то количество ссылок на объект остается равным 1, но только до того момента, как высвободится тот пул, в котором создан объект. При высвобождении всего пула объект также получает сообщение `release`. Если на данный момент количество ссылок на объект так и осталось равным 1, объект высвобождается.

В каждом потоке необходимо создавать автоматически высвобождаемый пул, причем это должен быть самый первый объект, создаваемый в конкретном потоке. Если этого не сделать, то любой объект, создаваемый в потоке на протяжении его существования, будет вызывать утечку памяти. Чтобы лучше понять эту проблему, рассмотрим следующий код:

```
- (void) autoreleaseThread:(id)paramSender{

  NSBundle *mainBundle = [NSBundle mainBundle];
  NSString *filePath = [mainBundle pathForResource:@"AnImage"
```

```
                                                ofType:@"png"];

  UIImage *image = [UIImage imageWithContentsOfFile:filePath];

  /* Делаем что-нибудь с изображением. */
  NSLog(@"Image = %@", image);

}

- (void)viewDidLoad {

  [super viewDidLoad];

  [NSThread detachNewThreadSelector:@selector(autoreleaseThread:)
                           toTarget:self
                         withObject:self];

}
```

Если запустить этот код и одновременно следить за окном консоли, то можно увидеть примерно следующее сообщение:

```
*** __NSAutoreleaseNoPool(): Object 0x5b2c990 of
class NSCFString autoreleased with no pool in place - just leaking
*** __NSAutoreleaseNoPool(): Object 0x5b2ca30 of
class NSPathStore2 autoreleased with no pool in place - just leaking
*** __NSAutoreleaseNoPool(): Object 0x5b205c0 of
class NSPathStore2 autoreleased with no pool in place - just leaking
*** __NSAutoreleaseNoPool(): Object 0x5b2d650 of
class UIImage autoreleased with no pool in place - just leaking
```

Эти данные свидетельствуют о том, что созданный нами автоматически высвобождаемый экземпляр `UIImage` приводит к утечке памяти. Более того, утечку вызывают и экземпляр класса `NSString` под названием `FilePath`, а также другие объекты, которые в обычной ситуации спокойно высвободились бы. Дело в том, что при создании потока мы забыли первым делом выделить и инициализировать автоматически высвобождаемый пул — именно первым делом. Далее приведен правильный код. Можете сами его протестировать и убедиться, что никаких утечек не возникает:

```
- (void) autoreleaseThread:(id)paramSender{

  @autoreleasepool {
    NSBundle *mainBundle = [NSBundle mainBundle];
    NSString *filePath = [mainBundle pathForResource:@"AnImage"
                                              ofType:@"png"];

    UIImage *image = [UIImage imageWithContentsOfFile:filePath];


    /* Делаем что-то с изображением. */
    NSLog(@"Image = %@", image);
```

```
  }

}
- (void)viewDidLoad {

[super viewDidLoad];
[NSThread detachNewThreadSelector:@selector(autoreleaseThread:)
toTarget:self
withObject:self];
}
```

# 7.16. Активизация фоновых методов

## Постановка задачи

Необходимо найти простой способ создания потоков так, чтобы с потоками не приходилось работать напрямую.

## Решение

Воспользуйтесь методом экземпляра `performSelectorInBackground:withObject:`, относящимся к классу `NSObject`:

```
- (BOOL)            application:(UIApplication *)application
  didFinishLaunchingWithOptions:(NSDictionary *)launchOptions{

  [self performSelectorInBackground:@selector(firstCounter)
                         withObject:nil];

  [self performSelectorInBackground:@selector(secondCounter)
                         withObject:nil];

  [self performSelectorInBackground:@selector(thirdCounter)
                         withObject:nil];

  self.window = [[UIWindow alloc] initWithFrame:
                 [[UIScreen mainScreen] bounds]];
  self.window.backgroundColor = [UIColor whiteColor];
  [self.window makeKeyAndVisible];
  return YES;
}
```

Методы счетчиков реализуются следующим образом:

```
- (void) firstCounter{

  @autoreleasepool {
    NSUInteger counter = 0;
    for (counter = 0;
```

```
        counter < 1000;
        counter++){
      NSLog(@"First Counter = %lu", (unsigned long)counter);
    }
  }

}

- (void) secondCounter{

  @autoreleasepool {
    NSUInteger counter = 0;
    for (counter = 0;
        counter < 1000;
        counter++){
      NSLog(@"Second Counter = %lu", (unsigned long)counter);
    }
  }

}

- (void) thirdCounter{

  @autoreleasepool {
    NSUInteger counter = 0;
    for (counter = 0;
        counter < 1000;
        counter++){
      NSLog(@"Third Counter = %lu", (unsigned long)counter);
    }
  }

}
```

## Обсуждение

Метод `performSelectorInBackground:withObject:` создает в фоновом режиме новый поток. Ситуация эквивалентна созданию нового потока для селекторов. Самое важное, что в данном случае нужно учитывать: поскольку этот метод создает поток для конкретного селектора, у селектора должен быть автоматически высвобождаемый пул, как и у любого другого потока, который действует в среде, управляемой с применением подсчета ссылок.

# 7.17. Выход из потоков и таймеров

## Постановка задачи

Требуется остановить поток или таймер либо не допустить его повторного запуска.

## Решение

При работе с таймерами пользуйтесь методом экземпляра `invalidate`, относящимся к классу `NSTimer`. При работе с потоками используйте метод `cancel`. Старайтесь не применять метод `exit` при работе с потоками, так как он не позволяет потоку произвести после себя очистку, что в итоге приводит к утечке ресурсов из вашего приложения.

```
NSThread *thread = /* Здесь получаем ссылку на ваш поток. */;
[thread cancel];

NSTimer *timer = /* Здесь получаем ссылку на ваш таймер. */;
[timer invalidate];
```

## Обсуждение

Выйти из таймера не составляет труда — можно просто вызвать метод экземпляра `invalidate`, относящийся к таймеру. После вызова этого метода таймер больше не будет инициировать никаких событий в своем целевом объекте.

А вот выходить из потоков немного сложнее. Когда поток находится в спящем режиме и вызывается его метод `cancel`, рабочий цикл этого потока выполнит свою задачу, а только потом осуществит выход. Рассмотрим это:

```
- (void) threadEntryPoint{

  @autoreleasepool {
    NSLog(@"Thread Entry Point");
    while ([[NSThread currentThread] isCancelled] == NO){
      [NSThread sleepForTimeInterval:4];
      NSLog(@"Thread Loop");
    }
    NSLog(@"Thread Finished");
  }

}

- (void) stopThread{

  NSLog(@"Cancelling the Thread");
  [self.myThread cancel];
  NSLog(@"Releasing the thread");
  self.myThread = nil;

}

- (BOOL)            application:(UIApplication *)application
  didFinishLaunchingWithOptions:(NSDictionary *)launchOptions{

  self.myThread = [[NSThread alloc]
                   initWithTarget:self
```

```
                    selector:@selector(threadEntryPoint)
                    object:nil];

  [self performSelector:@selector(stopThread)
             withObject:nil
             afterDelay:3.0f];

  [self.myThread start];

  self.window = [[UIWindow alloc] initWithFrame:
                 [[UIScreen mainScreen] bounds]];
  self.window.backgroundColor = [UIColor whiteColor];
  [self.window makeKeyAndVisible];
  return YES;
}
```

Данный код создает экземпляр класса `NSThread` и немедленно запускает поток. Поток в каждом цикле проводит 4 секунды в спящем режиме, а потом переходит к выполнению своей задачи. Тем не менее, прежде чем поток будет запущен, мы вызываем метод `stopThread`, относящийся к (написанному нами) контроллеру вида; это делается с трехсекундной задержкой. Данный метод вызывает метод `cancel`, относящийся к потоку, пытаясь заставить поток выйти из своего цикла. Теперь запустим приложение и посмотрим, что выводится в окне консоли:

```
...
Thread Entry Point
Cancelling the Thread
Releasing the thread
Thread Loop
Thread Finished
```

Итак, ясно видно, что перед выходом поток завершил текущий цикл, хотя запрос о выходе и был дан в середине цикла. Это очень распространенная ловушка. Чтобы избежать ее, нужно сначала проверять, не отменен ли поток, и лишь потом переходить к выполнению какой-либо задачи, для которой свойственны внешние побочные эффекты в цикле потока. Мы можем переписать код следующим образом. При этом операция с внешним эффектом (записыванием в регистрационный журнал) сначала проверяет, не отменен ли поток:

```
- (void) threadEntryPoint{

  @autoreleasepool {
    NSLog(@"Thread Entry Point");
    while ([[NSThread currentThread] isCancelled] == NO){
      [NSThread sleepForTimeInterval:4];
      if ([[NSThread currentThread] isCancelled] == NO){
        NSLog(@"Thread Loop");
      }
    }
    NSLog(@"Thread Finished");
```

```
  }

}

- (void) stopThread{
  NSLog(@"Cancelling the Thread");
  [self.myThread cancel];
  NSLog(@"Releasing the thread");
  self.myThread = nil;

}

- (BOOL)            application:(UIApplication *)application
  didFinishLaunchingWithOptions:(NSDictionary *)launchOptions{

  self.myThread = [[NSThread alloc]
                   initWithTarget:self
                   selector:@selector(threadEntryPoint)
                   object:nil];

  [self performSelector:@selector(stopThread)
             withObject:nil
             afterDelay:3.0f];

  [self.myThread start];

  self.window = [[UIWindow alloc] initWithFrame:
                 [[UIScreen mainScreen] bounds]];
  self.window.backgroundColor = [UIColor whiteColor];
  [self.window makeKeyAndVisible];
  return YES;
}
```

# 8 Безопасность

## 8.0. Введение

Безопасность — центральный элемент операционных систем iOS и OS X. Можно пользоваться функциями безопасности в iOS, чтобы без опасений хранить файлы на различных накопителях. Например, можно приказать iOS заблокировать и защитить на диске файлы с информацией из вашего приложения, если пользователь активизировал на устройстве защиту с применением пароля, а устройство перешло в режим блокировки. Если вы явно этого не затребуете, iOS не будет использовать с вашим приложением какого-либо защищенного хранилища данных. В таком случае ваши данные будут открыты для считывания любому процессу, имеющему доступ на считывание файловой системы вашего устройства. Существуют разнообразные приложения Mac, способные исследовать файловую систему устройства с iOS, не вызывая при этом джейлбрейка.

Джейлбрейк — это процесс предоставления доступа с правами администратора и снятия многих уровней защиты, действующих над операционной системой — например, над iOS. В частности, если устройство подверглось джейлбрейку, приложение может выполнить на нем неподписанный двоичный код. Но на обычном устройстве iOS приложение сможет быть выполнено на устройстве, лишь если оно имеет подпись Apple, полученную через App Store или верифицированный портал для разработки под iOS.

В Mac OS X Apple уже давно используется программа Keychain Access. Это инструмент, обеспечивающий пользователям iOS безопасное хранение данных на компьютере. Разработчики могут пользоваться программой Keychain Access, а также другими функциями обеспечения безопасности, выстроенными на базе общей архитектуры защиты данных (CDSA). Keychain Access может управлять различными связками ключей. Каждая связка ключей, в свою очередь, может содержать защищенные данные, например пароли. Если на машине с операционной системой OS X вы заходите под своим именем на сайт через браузер Safari, то вам будут предложены две возможности: приказать Safari запомнить ваш пароль либо отклонить этот запрос. Если вы прикажете Safari запомнить пароль, то браузер сохранит указанный пароль в заданной по умолчанию связке ключей.

Связки ключей в OS X и iOS различаются во многих отношениях. В OS X:

- у пользователя может быть много связок ключей, в iOS же действует всего одна глобальная связка ключей;

- пользователь может блокировать связку ключей. В iOS та связка ключей, которая используется по умолчанию, блокируется и разблокируется вместе с самим устройством;
- существует концепция стандартной связки ключей (default keychain), которая автоматически разблокируется операционной системой, когда пользователь входит в эту систему. Дело в том, что пароль стандартной связки ключей совпадает с паролем к пользовательской учетной записи. Как было указано ранее, в iOS есть всего одна связка ключей и она разблокируется iOS по умолчанию.

Чтобы помочь вам лучше понять природу связки ключей в OS X, прежде чем углубиться в изучение концепций iOS, связанных со связкой ключей и безопасностью, я хотел бы привести один пример. Откройте окно терминала на компьютере с Mac, введите следующую команду и нажмите Enter:

```
security list-keychains
```

В зависимости от настроек машины и имени пользователя вы получите примерно такой вывод:

```
"/Users/vandadnp/Library/Keychains/login.keychain"
"/Library/Keychains/System.keychain"
```

Как видите, здесь у меня две связки ключей: регистрационная и системная. Чтобы выяснить, какая связка ключей используется по умолчанию, введите следующую команду в окно терминала, а затем нажмите Enter:

```
security default-keychain
```

В типичной установке OS X эта команда вернет примерно такой результат:

```
"/Users/vandadnp/Library/Keychains/login.keychain"
```

Вывод свидетельствует, что у меня на машине по умолчанию используется регистрационная связка ключей. Поэтому по умолчанию все пароли, которые я приказываю запомнить различным программам в системе OS X, будут сохраняться именно в этой связке ключей, если конкретное приложение не решит, что пароль должен быть сохранен в другой связке ключей. Если нужной связки ключей пока не существует, приложению потребуется ее создать.

А теперь давайте попробуем кое-что интересное. Попытаемся узнать, какие пароли уже сохранены в нашей стандартной связке ключей. При этом будем исходить из того, что по умолчанию на машине используется связка ключей `login.keychain`, как мы определили ранее. Введите в окне терминала следующую команду и нажмите Enter:

```
security dump-keychain login.keychain | grep "password" -i
```

Аргумент `dump-keychain` для команды `security` в окне терминала дампирует все содержимое связки ключей в стандартный вывод. Для поиска паролей мы применили команду `grep`. Вывод этой команды может получиться примерно таким, как в следующем примере, в зависимости от того, какие пароли запомнил ваш компьютер:

```
"desc"<blob>="AirPort network password"
"desc"<blob>="AirPort network password"
"desc"<blob>="AirPort network password"
"desc"<blob>="AirPort network password"
"desc"<blob>="AirPort network password"
"desc"<blob>="AirPort network password"
"desc"<blob>="AirPort network password"
"desc"<blob>="AirPort network password"
"desc"<blob>="Web form password"
"desc"<blob>="Web form password"
"desc"<blob>="Web form password"
"desc"<blob>="Web form password"
```

Хорошо, все это очень интересно, но зачем я об этом рассказываю, как все это связано с iOS? Оказывается, что в архитектурном отношении связка ключей в iOS *очень* похожа на связку ключей в OS X, так как операционная система iOS создавалась на основе исходного кода OS X. Многие концепции iOS похожи на соответствующие элементы OS X, это касается и связки ключей. Необходимо отметить ряд очень важных аспектов, связанных со связками ключей в iOS, — в частности, поговорить о группах доступа и сервисах. Чтобы упростить вам изучение этой темы, я объясню, как эти концепции реализуются в OS X, а потом мы подробнее рассмотрим вариант связки ключей, применяемый в iOS.

На компьютере Mac нажмите клавиши Command + Пробел или просто нажмите значок Spotlight (Поиск) на верхней панели меню на экране (рис. 8.1).


**Рис. 8.1.** Нажмите значок Spotlight на панели меню для OS X

Когда откроется строка поиска, введите в нее запрос `Keychain Access` и нажмите Enter, чтобы открыть программу Keychain Access. В левой части окна этой программы находится область Keychains (Связки ключей). В ней выберите регистрационную связку ключей login, а затем в области Category (Категория) слева — запись Passwords (Пароли). Теперь перед вами должен открыться примерно такой интерфейс, как на рис. 8.2.

Keychain Access — это программа с графическим пользовательским интерфейсом, которая выстроена в OS X на базе интерфейсов API для работы со связкой ключей и обеспечения безопасности. Этот красивый и удобный интерфейс скрывает большую часть тех сложностей, которыми отличаются фреймворки безопасности в OS X. Теперь, если в системе есть какие-либо пароли, сохраненные в приложениях, например в Safari, вам потребуется дважды щелкнуть на одной из записей-паролей в правой части экрана. Откроется такое диалоговое окно, как на рис. 8.3.


**Рис. 8.2.** Программа Keychain Access в Mac OS X

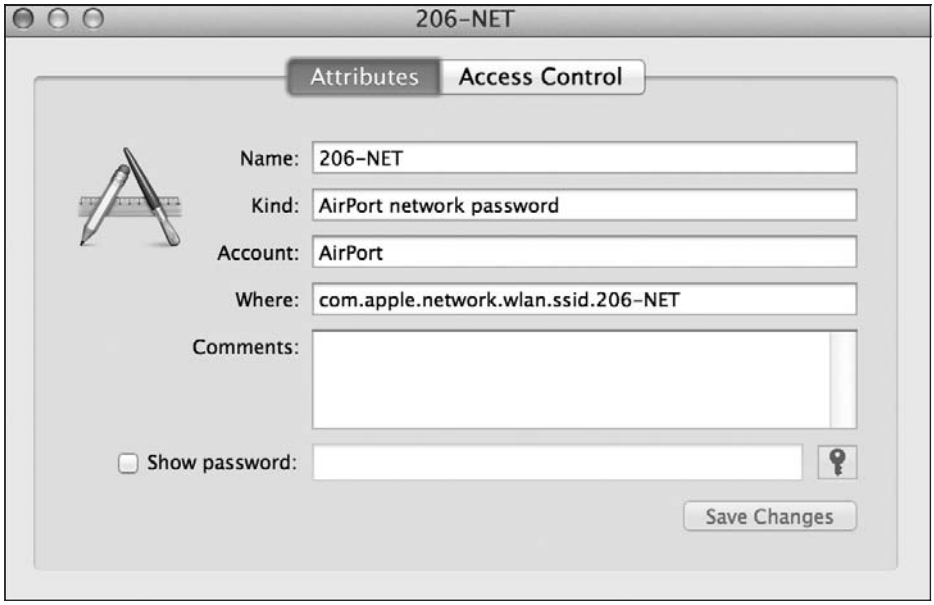


**Рис. 8.3.** Диалоговое окно Keychain Access, в котором отображается информация о сохраненном пароле

Познакомимся с некоторыми свойствами пароля, показанного на рис. 8.3.

- Name (Имя) — имя пароля, присвоенное ему приложением, сохранившим этот пароль. Так, в данном случае мы имеем дело с паролем для входа в беспроводную сеть 206-NET. Это имя иногда называют подписью.

- Kind (Вид) — вид элемента. В данном случае пароль относится к виду «пароль для беспроводных сетей». Это обычная строка, ее можно использовать для выполнения запросов к связке ключей (рассмотрено далее).
- Account (Учетная запись) — обычно это ключ для того значения, которое мы хотим сохранить. Связка ключей использует хранилище для пар «ключ/значение», так же как словарь из языка Objective-C. Ключ — это произвольная строка, и большинство приложений, сохраняющих элементы в связке ключей, хранят ключи и значения именно в этой секции.
- Where (местоположение) — часто именуется сервисом. Это идентификатор сервиса, сохранившего данный элемент в связке ключей. Этот идентификатор — как раз тот пароль, который вы запоминаете, и связка ключей фактически не имеет с ним дел, так как вы считаете этот пароль осмысленным. В iOS принято задавать в качестве имени этого сервиса идентификатор пакета нашего приложения. Так мы отличаем данные, сохраненные в конкретном приложении, от сохраненных данных из других приложений. Подробнее поговорим об этом в дальнейшем.

Кроме того, вы видите на рис. 8.3 флажок Show Password (Показать пароль). Если установить этот флажок, система будет запрашивать разрешение на показ пароля к конкретному элементу. Если вы введете свой пароль и дадите такое разрешение, то программа Keychain Access получит для вас защищенный пароль и отобразит его на экране.

Мы можем использовать команду `security` в окне терминала для выборки ровно той же информации. Если ввести в окне терминала следующую команду:

```
security find-generic-password -help
```

то получится примерно такой вывод:

```
Usage: find-generic-password [-a account] [-s service]
       [options...] [-g] [keychain...]
  -a Match "account" string
  -c Match "creator" (four-character code)
  -C Match "type" (four-character code)
  -D Match "kind" string
  -G Match "value" string (generic attribute)
  -j Match "comment" string
  -l Match "label" string
  -s Match "service" string
  -g Display the password for the item found
  -w Display only the password on stdout
If no keychains are specified to search, the default search list is used.
  Find a generic password item.
```

Итак, если передавать команде `security` требуемые параметры один за другим, то можно получить свойства интересующего вас пароля (см. рис. 8.3):

```
security find-generic-password
  -a "AirPort"
  -s "com.apple.network.wlan.ssid.206-NET"
```

```
-D "AirPort network password"
-l "206-NET"
-g   login.keychain
```

Как было показано ранее, команда `-g` запросит команду `security` отобразить пароль, ассоциированный с указанным элементом (при наличии такого пароля). Следовательно, после ввода этой команды в окно терминала вам будет предложено ввести пароль к вашей учетной записи перед продолжением работы. Аналогичным образом мы указывали пароль к учетной записи, чтобы отобразить пароль на рис. 8.3.

В iOS, притом что во всей операционной системе применяется единая глобальная область действия связки ключей, приложение все равно может считывать и записывать информацию лишь на небольшом экранированном участке связки ключей (по принципу работы в песочнице). Два приложения, написанные одним и тем же разработчиком (и подписанные одинаковым профилем инициализации с одного и того же портала для разработки в iOS), могут получать доступ к общей разделяемой области связки ключей, но при этом сохраняют и собственный ограниченный доступ каждый к своей связке ключей. Итак, два приложения, App X и App Y, написанные одним и тем же iOS-разработчиком, могут получить доступ к следующим областям связки ключей.

- App X — к области связки ключей приложения App X.
- App Y — к области связки ключей приложения App Y.
- Приложения App X и App Y обладают доступом к общему разделяемому участку связки ключей (с применением групп доступа, если программист соответствующим образом сконфигурирует разрешения для этого приложения).
- App X не может считывать информацию из связки ключей приложения App Y, а приложение App Y не может считывать аналогичные данные приложения App X.

Операционная система iOS просматривает *разрешения* приложения, чтобы определить требуемый тип доступа. Разрешения для конкретной программы кодируются в профиле инициализации, который использовался при подписывании приложения. Допустим, мы создали новый профиль инициализации под названием `KeychainTest_Dev.mobileprovision`. Этот профиль был размещен на Рабочем столе. С помощью следующей команды можно извлечь разрешения, соответствующие данному профилю, вот так:

```
cd ~/Desktop
```

Эта команда выведет вас на Рабочий стол. Уже здесь нужно выполнить следующую команду, чтобы считать разрешения из профиля инициализации:

```
security cms -D -i KeychainTest_Dev.mobileprovision | grep -A12 "Entitlements"
```

Показанная здесь команда security декодирует весь профиль инициализации, после чего команда grep просмотрит раздел Entitlements (Разрешения) в профиле и считает 12 строк текста от начала этого раздела. Если в вашем разделе с разрешениями содержится больше или меньше текста, то нужно соответствующим образом изменить аргумент -A12.

Вывод этой команды будет выглядеть примерно следующим образом в зависимости от вашего профиля:

```
<key>Entitlements</key>
<dict>
  <key>application-identifier</key>
  <string>F3FU372W5M.com.pixolity.ios.cookbook.KeychainTest</string>
  <key>com.apple.developer.default-data-protection</key>
  <string>NSFileProtectionComplete</string>
  <key>get-task-allow</key>
  <true/>
  <key>keychain-access-groups</key>
  <array>
    <string>F3FU372W5M.*</string>
  </array>
</dict>
```

Самый важный раздел, который нас здесь интересует, называется `keychain-access-groups`. Здесь указываются группы доступа к нашим элементам из связки ключей. Это групповой идентификатор для всех приложений, разработанных одним и тем же программистом. Здесь `F3FU372W5M` — это мой командный идентификатор с портала разработки для iOS. Идущий далее астериск показывает, в какие группы доступа я могу позже поместить те элементы, которые требуется хранить в защищенном режиме. В данном случае астериск означает *любую группу*. Поэтому по умолчанию это приложение получает доступ к элементам связок ключей любого приложения, которое разработано членами вышеупомянутой команды. Не волнуйтесь, если пока не совсем улавливаете суть. В этой главе вы подробно изучите тему безопасности в iOS, в частности, к концу главы будете знать все необходимое об использовании связок ключей в iOS.

Совершенно необходимо добавить к вашему приложению фреймворк Security (Безопасность), перед тем как переходить к изучению разделов этой главы. Большинство приемов, описанных в данной главе, работает со службами связки ключей в iOS, а для этого необходимо наличие фреймворка Security. В iOS SDK 7 появилась концепция модулей. Поэтому если вы просто импортируете обобщающий заголовок фреймворка безопасности в ваш проект, LLVM сам свяжет ваше приложение с нужным модулем безопасности — делать это вручную не придется. Вам всего лишь понадобится гарантировать, что в настройках сборки активизирована функция Enable Modules (Использовать модули), а также импортировать в проект следующий заголовок:

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

В Xcode 5 также появилась поддержка «возможностей» (Capabilities). Это новая вкладка, расположенная рядом с вкладкой Build Settings (Настройки сборки). Здесь вы можете с легкостью добавлять разрешения к вашему приложению и даже активизировать связку ключей без особых хлопот. Тем не менее, работая таким образом, вы теряете доступ практически ко всем деталям и не можете создавать собственные профили инициализации. В вашем распоряжении будут только джокерные профили инициализации, которыми мы обычно не пользуемся, реализуя в приложе-

ниях пуш-уведомления и другие возможности. Рекомендую хотя бы познакомиться с этой новой вкладкой. Просто щелкните на файле с вашим проектом в Xcode, посмотрите на правую часть экрана и выберите Capabilities (Возможности). Затем можно будет отключить компоненты iCloud и Keychain Access.

# 8.1. Обеспечение безопасности и защиты в приложениях

## Постановка задачи

Требуется сохранять значения в связке ключей и обеспечить в приложении безопасное хранение данных.

## Решение

Создайте в приложении профиль инициализации, в котором активизирована защита файлов.

## Обсуждение

Профили инициализации, уже упоминавшиеся ранее (см. раздел 8.0), содержат разрешения. Разрешения указывают системе iOS, как ваше приложение использует возможности безопасности, предоставляемые операционной системой. В эмуляторе iOS используется код без подписи (отсутствует этап, называемый code signing), следовательно, такие концепции будут лишены смысла. Но при отладке приложения или отправке его в App Store вы должны будете гарантировать, что приложение подписано корректным профилем инициализации одновременно для схем Debug (Отладка) и Release (Выпуск).

Я продемонстрирую все шаги, необходимые для создания действительного профиля реализации при разработке, а также опишу профили Ad Hoc и App Store. Выполните следующие шаги, чтобы создать валидный профиль инициализации для целей разработки (с поддержкой отладки). Он будет использоваться с приложениями, над которыми мы будем работать в данной главе. Начнем с создания ID приложения.

Предполагается, что вы уже создали для себя валидные сертификаты на разработку и распространение приложения.

1. Перейдите в iOS Dev Center и войдите в систему под своими именем и паролем.
2. Найдите раздел iOS Development Program (Программа разработки для iOS) и выберите Certificates, Identifiers & Profiles (Сертификаты, идентификаторы и профили).

3. Найдите в левой части экрана область App IDs (Идентификаторы приложений) и нажмите кнопку +, чтобы создать новый такой идентификатор.
4. В области Name (Имя) введите имя `Security App`. На самом деле здесь можно указать любое имя, но, чтобы избежать путаницы в этой главе, лучше воспользоваться данным именем, которое я буду применять в примерах.
5. В области App Services (Сервисы приложения) установите флажок Data Protection (Защита данных) и убедитесь, что выбран параметр Complete Protection (Полная защита). Остальные настройки оставьте без изменений.
6. Убедитесь, что в области App ID Suffix (Суффикс идентификатора приложения) установлен флажок Explicit App ID (Явный идентификатор приложения), и введите в поле Bundle ID (Идентификатор пакета) имя сервиса с точкой в качестве разделителя. Рекомендую имя `com.NAME.ios.cookbook.SecurityApp`, где `NAME` — название вашей компании. Если у вас нет компании с названием, просто придумайте его! В примерах я использую название `com.pixolity.ios.cookbook.SecurityApp`, но вам потребуется уникальное имя, это вы не сможете использовать.
7. Выполнив эти шаги, нажмите кнопку Continue (Продолжить).
8. Теперь система запросит у вас подтверждения заданных настроек, прежде чем будет создан идентификатор приложения. Вы увидите картинку, примерно как на рис. 8.4.
9. Когда закончите подготовку настроек, нажмите кнопку Submit (Отправить), чтобы создать ID приложения.

Прекрасно! Теперь у нас есть идентификатор приложения, но еще требуется создать профили инициализации. Я подробно расскажу, как создается профиль инициализации для разработки, а создание профилей Ad Hoc и App Store оставляю вам в качестве самостоятельной работы, так как процесс практически идентичен. Выполните следующие шаги, чтобы создать профиль инициализации для целей разработки.

1. В области Certificates, Identifiers & Profiles (Сертификаты, идентификаторы и профили) портала разработки выберите область Development (Разработка) в категории Provisioning Profiles (Профили инициализации). Затем нажмите кнопку +.
2. В открывшемся окне в области Development (Разработка) установите флажок iOS App Development (Разработка приложения для iOS) и нажмите кнопку Continue (Продолжить).
3. Когда система потребует выбрать идентификатор приложения (App ID), выберите тот App ID, который вы создали ранее. В моем случае это будет App ID, показанный на рис. 8.5. Сделав выбор, нажмите кнопку Continue (Продолжить).
4. Выберите сертификат (-ы) разработки, с которыми хотите связать ваш профиль. Затем нажмите кнопку Continue (Продолжить).
5. Выберите список устройств, на которые можно будет установить ваш профиль (это делается только для профилей Development и Ad Hoc, но не для App Store) и нажмите кнопку Continue (Продолжить).


**Рис. 8.4.** Подтверждение настроек App ID перед созданием идентификатора приложения

6. На следующем экране система потребует от вас указать имя вашего профиля. Введите имя, соответствующее правилам из Security App Dev Profile, а потом нажмите кнопку Generate (Сгенерировать), чтобы создать профиль инициализации.
7. Теперь ваш профиль готов к загрузке (рис. 8.6). Нажмите кнопку Download (Загрузить), чтобы загрузить профиль.
8. Для установки профиля перетащите загруженный профиль в iTunes. В результате профиль с его оригинальным именем будет установлен в каталоге ~/Library/MobileDevice/Provisioning Profiles/. Мне известно, что многие iOS-разработчики устанавливают профиль инициализации, просто дважды щелкнув на нем кнопкой мыши. Такой способ частично работает, то есть при двойном щелчке профиль действительно устанавливается в упомянутом каталоге. Но оригинальное имя профиля при этом стирается и заменяется SHA1-хешем профиля. Если позже вы откроете каталог, то не сможете понять, какой профиль вам нужен. Придется просматривать все профили и выяснять их имена. Поэтому я настоятельно не рекомендую устанавливать профили двойным щелчком кнопкой мыши. Лучше перетаскивать их в iTunes или вручную вставлять в данный каталог.

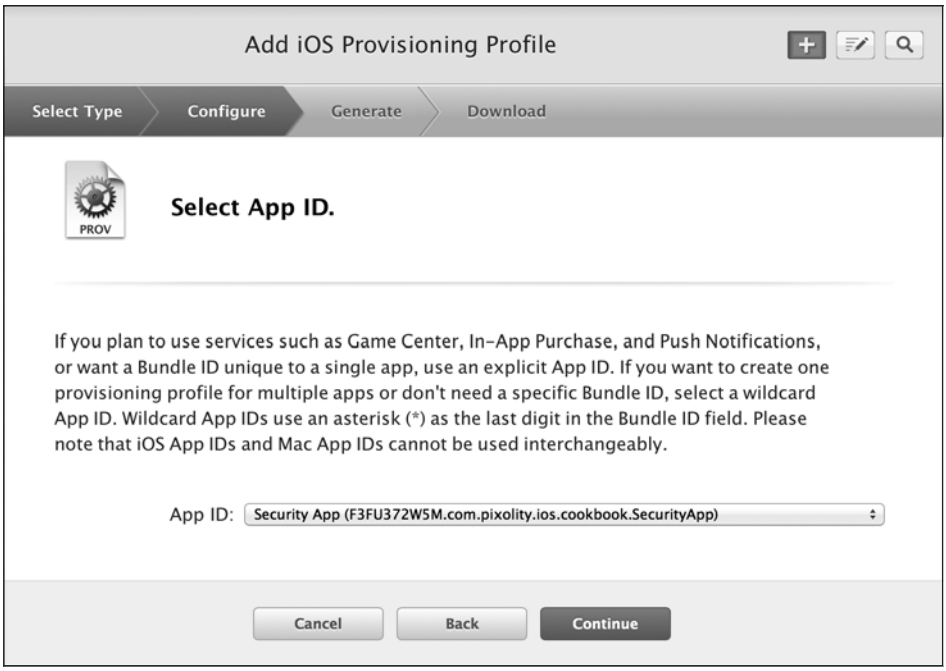


**Рис. 8.5.** Выбор нового идентификатора приложения для нового профиля инициализации (для целей разработки)

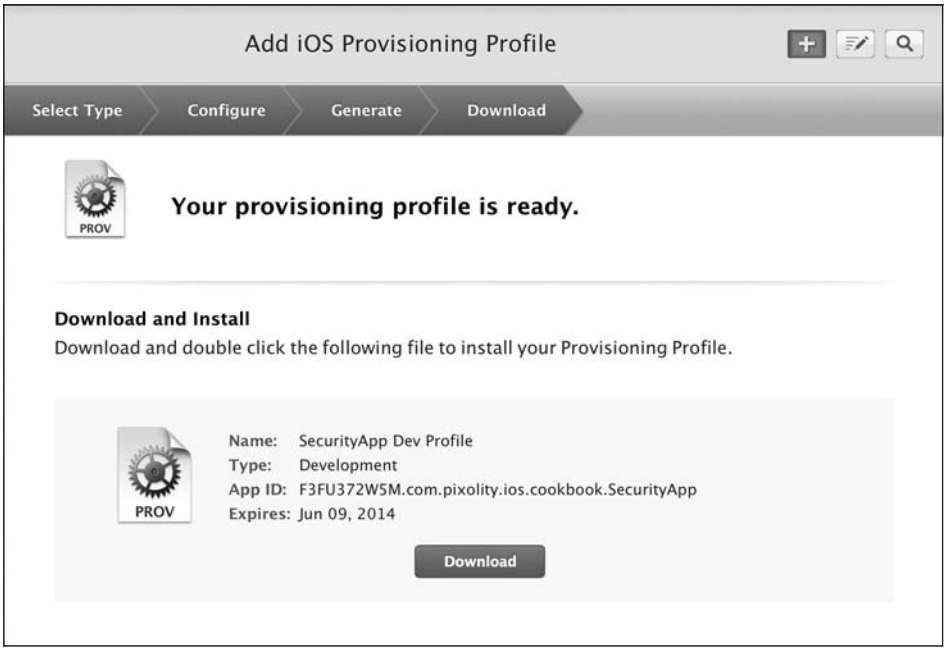


**Рис. 8.6.** Профиль для разработки сгенерирован и готов к загрузке

Блестяще. Вы установили на компьютере профиль инициализации для вашего приложения. Воспользуйтесь настройками сборки проекта, чтобы убедиться в том, что выбран правильный профиль для схемы Debug (Отладка). Затем повторите описанный процесс при создании профилей Ad Hoc и App Store, чтобы гарантировать сборку приложения с нужными профилями для схемы Release (Выпуск).

Созданный вами профиль инициализации обеспечивает отладку ваших приложений на устройстве с iOS и удобное сохранение данных на диске с применением связки ключей.

## См. также

Раздел 8.0.

# 8.2. Хранение значений в связке ключей

## Постановка задачи

Требуется обеспечить безопасное хранение конфиденциальных данных в связке ключей.

## Решение

Необходимо гарантировать, что ваше приложение будет скомпоновано с учетом требований фреймворка Security (Безопасность). Затем воспользуйтесь функцией `SecItemAdd` для добавления нового элемента в связку ключей приложения.

## Обсуждение

API связки ключей в операционных системах iOS и OS X написаны на языке C. Таким образом, у нас нет мостика к Objective-C или какого-то промежуточного уровня, который предоставлял бы взаимодействие программы с API на C. Поэтому работать с этими API несколько сложнее, чем с обычными. Основной момент при изучении этих API заключается в том, что запросы, отправляемые к API связки ключей, обычно упакованы в словарях. Например, если требуется запросить у сервисов связки ключей безопасное хранение тех или иных данных, то вы помещаете этот запрос в словарь (а вместе с запросом — все данные, которые собираетесь хранить, ключ к этим данным, идентификатор приложения и т. д.). Этот словарь вы отправляете к API, примером которого может служить функция `SecItemAdd`. Чтобы хранить информационный фрагмент в связке ключей, создайте словарь со следующими ключами:

- `kSecClass` — при необходимости хранения конфиденциальных информационных фрагментов, например строк, в качестве значения этого ключа обычно задается `kSecClassGenericPassword`;

- kSecAttrService — значение ключа чаще всего представляет собой строку. Как правило, эта строка — идентификатор нашего приложения;
- kSecAttrAccount — значением является строка, указывающая ключ к значению, которое мы хотим сохранить. Это произвольная строка, которая должна иметь смысл для вас и в контексте приложения;
- kSecValueData — значением является экземпляр NSData, который вы хотите сохранить по указанному ключу (kSecAttrAccount).

Возвращаемое значение функции SecItemAdd относится к типу OSStatus. Различные значения, которые вы можете получить от этой функции, определяются в файле SecBase.h SDK. Поэтому, находясь в Xcode, просто нажмите комбинацию клавиш Command+Shift+O, введите SecBase.h и попробуйте найти значение errSecSuccess. После того как найдете errSecSuccess в перечне, вы сможете просмотреть остальные значения, которые могут быть возвращены в экземпляре OSStatus:

```
enum
{
  errSecSuccess                 = 0,
  errSecUnimplemented           = -4,
  errSecParam                   = -50,
  errSecAllocate                = -108,
  errSecNotAvailable            = -25291,
  errSecDuplicateItem           = -25299,
  errSecItemNotFound            = -25300,
  errSecInteractionNotAllowed = -25308,
  errSecDecode                  = -26275,
  errSecAuthFailed              = -25293,
};
```

Если функция SecItemAdd завершится успешно, то в качестве ее возвращаемого значения вы получите errSecSuccess. Иное значение, получаемое от этой функции, означает ошибку. Итак, давайте объединим изученное и сделаем небольшой код, который будет записывать строковое значение в связку ключей:

```
#import "AppDelegate.h"
#import <Security/Security.h>

@implementation AppDelegate

- (BOOL)            application:(UIApplication *)application
  didFinishLaunchingWithOptions:(NSDictionary *)launchOptions{

    NSString *key = @"Full Name";
    NSString *value = @"Steve Jobs";
    NSData *valueData = [value dataUsingEncoding:NSUTF8StringEncoding];
    NSString *service = [[NSBundle mainBundle] bundleIdentifier];
    NSDictionary *secItem = @{
      (__bridge id)kSecClass : (__bridge id)kSecClassGenericPassword,
      (__bridge id)kSecAttrService : service,
      (__bridge id)kSecAttrAccount : key,
```

```
      (__bridge id)kSecValueData : valueData,
    };

    CFTypeRef result = NULL;
    OSStatus status = SecItemAdd((__bridge CFDictionaryRef)secItem, &result);

    if (status == errSecSuccess){
      NSLog(@"Successfully stored the value");
    } else {
      NSLog(@"Failed to store the value with code: %ld", (long)status);
    }

    self.window = [[UIWindow alloc]
    initWithFrame:[[UIScreen mainScreen] bounds]];
    self.window.backgroundColor = [UIColor whiteColor];
    [self.window makeKeyAndVisible];
    return YES;
}
```

При первом запуске этого приложения (предполагается, что вы выполнили все рекомендации из предыдущих разделов этой главы и правильно настроили свой профиль) вы получите от функции `SecItemAdd` значение `errSecSuccess`. Однако если вы повторно запустите это приложение, то получите значение `errSecDuplicateItem`. Таким образом iOS сообщает вам, что нельзя перезаписать имеющееся значение. В контексте строгих требований безопасности, применяемых в связке ключей, перезаписать имеющееся значение действительно нельзя. Но вот *обновить* имеющееся значение вы можете. О том, как это делается, рассказано далее в данной главе.

## См. также

Раздел 8.1.

# 8.3. Нахождение значений в связке ключей

## Постановка задачи

Требуется найти в связке ключей имеющийся там элемент.

## Решение

Воспользуйтесь функцией `SecItemCopyMatching`. Выполните следующие шаги.

1. Создайте словарь для передачи вышеупомянутой функции. Добавьте к словарю ключ `kSecClass`. Присвойте ключу такое значение, чтобы он отражал тип искомого элемента. Как правило, здесь требуется значение `kSecClassGenericPassword`.
2. Добавьте в словарь ключ `kSecAttrService`. В качестве значения этого ключа установите сервисную строку того элемента, который вы ищете. В этой главе в качестве

имен сервисов мы используем идентификатор пакета нашего приложения и устанавливаем в качестве идентификаторов пакета всех приложений одну и ту же строку. Таким образом, одно приложение может считывать связку ключей, другое — записывать в нее данные и т. д.

3. Добавьте в словарь ключ `kSecAttrAccount` и задайте в качестве его значения текущий ключ того значения, которое вы ранее сохранили в связке ключей. Если вы внимательно проработали пример из раздела 8.2, то в данном случае именем учетной записи будет служить строка `Full Name`.
4. Добавьте к словарю атрибут `kSecReturnAttributes`. В качестве его значения задайте `kCFBooleanTrue`, если хотите получить атрибуты значения, присутствующего в связке ключей (например, дату создания и дату последнего изменения). Если хотите получить текущее значение элемента, сохраненного вами в связке ключей, а не ключ `kSecReturnAttributes`, добавьте к словарю ключ `kSecReturnData` и задайте для него значение `kCFBooleanTrue`.

Как только ваш словарь будет готов, вы сможете передать его функции `SecItemCopyMatching` в качестве первого параметра. Второй параметр — это указатель на тот объект, который будет возвращаться функцией. Этот указатель должен относиться к типу `CFTypeRef *`. Это обобщенный тип данных, и в каждом конкретном случае тип зависит от того, что именно вы передадите функции `SecItemCopyMatching` в качестве первого параметра. Например, если в вашем словаре содержится ключ `kSecReturnAttributes`, то вторым параметром этой функции должен быть либо `NULL`, либо указатель на непрозрачный тип `CFDictionaryRef`. Если же вместо этого вы передадите словарю ключ `kSecReturnData`, то второй параметр этой функции должен относиться к типу `CFDataRef`. Это непрозрачный тип, который будет получать точные данные имеющегося элемента. Затем вы сможете преобразовать эти данные в экземпляр `NSString` и работать с ним.

## Обсуждение

Предположим, вы хотите считать *свойства* той строки, которую записали в связку ключей в разделе 8.2. Можно написать такой код:

```
- (BOOL)            application:(UIApplication *)application
  didFinishLaunchingWithOptions:(NSDictionary *)launchOptions{

    NSString *keyToSearchFor = @"Full Name";
    NSString *service = [[NSBundle mainBundle] bundleIdentifier];

    NSDictionary *query = @{
      (__bridge id)kSecClass : (__bridge id)kSecClassGenericPassword,
      (__bridge id)kSecAttrService : service,
      (__bridge id)kSecAttrAccount : keyToSearchFor,
      (__bridge id)kSecReturnAttributes : (__bridge id)kCFBooleanTrue,
    };

    CFDictionaryRef valueAttributes = NULL;
    OSStatus results = SecItemCopyMatching((__bridge CFDictionaryRef)query,
```

```
                                          (CFTypeRef *)&valueAttributes);

    NSDictionary *attributes =
      (__bridge_transfer NSDictionary *)valueAttributes;

    if (results == errSecSuccess){

      NSString *key, *accessGroup, *creationDate, *modifiedDate, *service;

      key = attributes[(__bridge id)kSecAttrAccount];
      accessGroup = attributes[(__bridge id)kSecAttrAccessGroup];
      creationDate = attributes[(__bridge id)kSecAttrCreationDate];
      modifiedDate = attributes[(__bridge id)kSecAttrModificationDate];
      service = attributes[(__bridge id)kSecAttrService];

      NSLog(@"Key = %@\n \
            Access Group = %@\n \
            Creation Date = %@\n \
            Modification Date = %@\n \
            Service = %@", key, accessGroup, creationDate,
            modifiedDate, service);

    } else {
      NSLog(@"Error happened with code: %ld", (long)results);
    }

    self.window = [[UIWindow alloc]
                   initWithFrame:[[UIScreen mainScreen] bounds]];
    self.window.backgroundColor = [UIColor whiteColor];
    [self.window makeKeyAndVisible];
    return YES;
}
```

После запуска этого приложения на консоль будут выведены примерно такие результаты:

```
Key = Full Name
Access Group = F3FU372W5M.com.pixolity.ios.cookbook.SecurityApp
Creation Date = 2013-06-0910:44:55 +0000
Modification Date = 2013-06-0910:44:55 +0000
Service = com.pixolity.ios.cookbook.SecurityApp
```

Это, конечно, хорошо, но как считывать актуальную информацию о значении? В подразделе «Решение» данного раздела я уже ответил на этот вопрос: необходимо включить в запрос ключ `kSecReturnData`. После того как это будет сделано, в качестве второго параметра функция `SecItemCopyMatching` потребует либо `NULL`, либо непрозрачную переменную `CFDataRef`, вот так:

```
#import "AppDelegate.h"
#import <Security/Security.h>

@implementation AppDelegate
```

```
- (BOOL) application:(UIApplication *)application
  didFinishLaunchingWithOptions:(NSDictionary *)launchOptions{

    NSString *keyToSearchFor = @"Full Name";
    NSString *service = [[NSBundle mainBundle] bundleIdentifier];

    NSDictionary *query = @{
      (__bridge id)kSecClass : (__bridge id)kSecClassGenericPassword,
      (__bridge id)kSecAttrService : service,
      (__bridge id)kSecAttrAccount : keyToSearchFor,
      (__bridge id)kSecReturnData : (__bridge id)kCFBooleanTrue,
    };

    CFDataRef cfValue = NULL;
    OSStatus results = SecItemCopyMatching((__bridge CFDictionaryRef)query,
                                           (CFTypeRef *)&cfValue);

    if (results == errSecSuccess){

      NSString *value = [[NSString alloc]
                         initWithData:(__bridge_transfer NSData *)cfValue
                         encoding:NSUTF8StringEncoding];

      NSLog(@"Value = %@", value);

    } else {
      NSLog(@"Error happened with code: %ld", (long)results);
    }

    self.window = [[UIWindow alloc]
    initWithFrame:[[UIScreen mainScreen] bounds]];
    self.window.backgroundColor = [UIColor whiteColor];
    [self.window makeKeyAndVisible];
    return YES;
}
```

По умолчанию функция `SecItemCopyMatching` ищет первое совпадение в связке ключей. Допустим, вы сохранили в связке ключей 10 безопасных элементов класса `kSecClassGenericPassword` и хотите запросить их все. Как это сделать? Ничего сложного. Просто добавьте ключ `kSecMatchLimit` в словарь вашего запроса и укажите максимальное количество совпадающих элементов, которые сервисы должны отыскивать в связке ключей. Можно также присвоить этому ключу значение `kSecMatchLimitAll` — при нем осуществляется поиск всех совпадений. Когда вы внедрите ключ `kSecMatchLimit` в ваш словарь запроса для функции `SecItemCopyMatching`, второй параметр этого типа обязательно потребует указатель на непрозрачный тип `CFArrayRef`, а состоять этот массив будет только из тех элементов, которые вы запросили.

Рассмотрим пример, в котором мы пытаемся найти *все* элементы связки ключей, удовлетворяющие определенным критериям:

```
#import "AppDelegate.h"
#import <Security/Security.h>

@implementation AppDelegate

- (BOOL) application:(UIApplication *)application
  didFinishLaunchingWithOptions:(NSDictionary *)launchOptions{

    NSString *keyToSearchFor = @"Full Name";
    NSString *service = [[NSBundle mainBundle] bundleIdentifier];

    NSDictionary *query = @{
      (__bridge id)kSecClass : (__bridge id)kSecClassGenericPassword,
      (__bridge id)kSecAttrService : service,
      (__bridge id)kSecAttrAccount : keyToSearchFor,
      (__bridge id)kSecReturnData : (__bridge id)kCFBooleanTrue,
      (__bridge id)kSecMatchLimit : (__bridge id)kSecMatchLimitAll
    };

    CFArrayRef allCfMatches = NULL;
    OSStatus results = SecItemCopyMatching((__bridge CFDictionaryRef)query,
                                           (CFTypeRef *)&allCfMatches);

    if (results == errSecSuccess){

      NSArray *allMatches = (__bridge_transfer NSArray *)allCfMatches;

      for (NSData *itemData in allMatches){
        NSString *value = [[NSString alloc]
                           initWithData:itemData
                           encoding:NSUTF8StringEncoding];
        NSLog(@"Value = %@", value);
      }
    } else {
      NSLog(@"Error happened with code: %ld", (long)results);
    }

    self.window = [[UIWindow alloc]
    initWithFrame:[[UIScreen mainScreen] bounds]];
    self.window.backgroundColor = [UIColor whiteColor];
    [self.window makeKeyAndVisible];
    return YES;
}
```

## См. также

Раздел 8.2.

# 8.4. Обновление значений в связке ключей

## Постановка задачи

Вы уже сохранили значение в связке ключей, а теперь хотите обновить его.

## Решение

Учитывая, что вы смогли найти значение в связке ключей (см. раздел 8.3), можно схожим образом выполнить функцию `SecItemUpdate` с двумя параметрами. Ее первым параметром будет словарь вашего запроса, а вторым — словарь, описывающий изменения, которые вы хотите внести в имеющееся значение. Обычно этот обновляющий словарь (второй параметр метода) содержит всего один ключ (`kSecValueData`). Значением этого словарного ключа являются данные, которые нужно заново установить для имеющегося в связке ключа.

## Обсуждение

Допустим, вы выполнили все указания, изложенные в разделе 8.2, и сохранили в связке ключей приложения строку `Steve Jobs` с ключом `Full Name`. Теперь вы хотите обновить это значение. Первым делом понадобится определить, присутствует ли уже нужное значение в связке ключей. Для этого создадим простой запрос, который вы уже видели ранее в этой главе:

```
NSString *keyToSearchFor = @"Full Name";
NSString *service = [[NSBundle mainBundle] bundleIdentifier];

NSDictionary *query = @{
  (__bridge id)kSecClass : (__bridge id)kSecClassGenericPassword,
  (__bridge id)kSecAttrService : service,
  (__bridge id)kSecAttrAccount : keyToSearchFor,
};
```

Затем сделаем запрос к этому словарю и проверим, находится ли интересующий нас элемент в связке ключей:

```
OSStatus found = SecItemCopyMatching((__bridge CFDictionaryRef)query,
                                     NULL);
```

Можно и не проверять наличие значения перед тем, как его обновлять. Вполне допустимо просто попытаться обновить значение. Если же его не существует, функция SecItemUpdate вернет значение errSecItemNotFound. Выбор заключается в том, проводить ли поиск в связке ключей самостоятельно или перепоручить эту задачу SecItemUpdate.

Если эта функция вернет значение `errSecSuccess`, вы будете знать, что интересовавшее вас значение уже обновлено. Обратите внимание: в качестве второго параметра мы передали `NULL`. Дело в том, что мы не собираемся получать из

связки ключей старое значение. Мы просто хотим определить, существует ли значение, а сделать это можем, только проверив возвращаемое значение функции. Если возвращаемое значение равно errSecSuccess, делаем вывод, что значение уже было сохранено и может быть обновлено. Обновлять значение мы будем вот так:

```
NSData *newData = [@"Mark Tremonti"
                   dataUsingEncoding:NSUTF8StringEncoding];

NSDictionary *update = @{
                          (__bridge id)kSecValueData : newData,
                         };

OSStatus updated = SecItemUpdate((__bridge CFDictionaryRef)query,
                                 (__bridge CFDictionaryRef)update);

if (updated == errSecSuccess){
  NSLog(@"Successfully updated the existing value");
} else {
  NSLog(@"Failed to update the value. Error = %ld", (long)updated);
}
```

Обновляющий словарь, который мы передаем функции SecItemUpdate в качестве второго параметра, может содержать больше ключей чем один ключ kSecValueData, использованный в нашем примере. На самом деле этот словарь может содержать обновления для любого имеющегося элемента. Например, если вы хотите добавить комментарий к имеющемуся значению (комментарий — это строка), то можете выполнить обновление следующим образом:

```
#import "AppDelegate.h"
#import <Security/Security.h>

@implementation AppDelegate

- (void) readExistingValue{

  NSString *keyToSearchFor = @"Full Name";
  NSString *service = [[NSBundle mainBundle] bundleIdentifier];

  NSDictionary *query = @{
    (__bridge id)kSecClass : (__bridge id)kSecClassGenericPassword,
    (__bridge id)kSecAttrService : service,
    (__bridge id)kSecAttrAccount : keyToSearchFor,
    (__bridge id)kSecReturnAttributes : (__bridge id)kCFBooleanTrue,
  };

  CFDictionaryRef cfAttributes = NULL;
  OSStatus found = SecItemCopyMatching((__bridge CFDictionaryRef)query,
                                       (CFTypeRef *)&cfAttributes);

  if (found == errSecSuccess){
```

```
    NSDictionary *attributes =
      (__bridge_transfer NSDictionary *)cfAttributes;

    NSString *comments = attributes[(__bridge id)kSecAttrComment];
    NSLog(@"Comments = %@", comments);

  } else {
    NSLog(@"Error happened with code: %ld", (long)found);
  }

}

- (BOOL) application:(UIApplication *)application
  didFinishLaunchingWithOptions:(NSDictionary *)launchOptions{

    NSString *keyToSearchFor = @"Full Name";
    NSString *service = [[NSBundle mainBundle] bundleIdentifier];

    NSDictionary *query = @{
      (__bridge id)kSecClass : (__bridge id)kSecClassGenericPassword,
      (__bridge id)kSecAttrService : service,
      (__bridge id)kSecAttrAccount : keyToSearchFor,
    };

    OSStatus found = SecItemCopyMatching((__bridge CFDictionaryRef)query,
                                         NULL);

    if (found == errSecSuccess){

      NSData *newData = [@"Mark Tremonti"
      dataUsingEncoding:NSUTF8StringEncoding];

      NSDictionary *update = @{
        (__bridge id)kSecValueData : newData,
        (__bridge id)kSecAttrComment : @"My Comments",
      };

      OSStatus updated = SecItemUpdate((__bridge CFDictionaryRef)query,
                                       (__bridge CFDictionaryRef)update);

      if (updated == errSecSuccess){
        [self readExistingValue];

      } else {
        NSLog(@"Failed to update the value. Error = %ld", (long)updated);
      }
    } else {
      NSLog(@"Error happened with code: %ld", (long)found);
    }
```

```
    self.window = [[UIWindow alloc]
                   initWithFrame:[[UIScreen mainScreen] bounds]];
    self.window.backgroundColor = [UIColor whiteColor];
    [self.window makeKeyAndVisible];
    return YES;
}
```

В этом примере важнее всего отметить, что мы включили в обновляющий словарь ключ `kSecAttrComment`. Как только обновление будет выполнено, мы считаем комментарий с помощью того самого метода считывания, который изучили в разделе 8.3.

## См. также

Разделы 8.2 и 8.3.

# 8.5. Удаление значений из связки ключей

## Постановка задачи

Требуется удалить элемент из связки ключей.

## Решение

Воспользуйтесь функцией `SecItemDelete`.

## Обсуждение

В разделе 8.2 мы научились сохранять значения в связке ключей. Для удаления этих значений потребуется использовать функцию `SecItemDelete`. Эта функция принимает всего один параметр: словарь типа `CFDictionaryRef`. Можно взять обычный словарь и преобразовать его в экземпляр `CFDictionaryRef` с помощью мостика, как мы поступали в других разделах этой главы. Словарь, передаваемый этому методу, должен содержать следующие ключи:

- `kSecClass` — тип элемента, который вы собираетесь удалить, например `kSecClassGenericPassword`;
- `kSecAttrService` — сервис, к которому привязан элемент. Сохраняя элемент, вы подбираете для него сервис, и этот же сервис вы должны указать здесь. Так, в предыдущих примерах мы задавали в качестве значения этого ключа идентификатор пакета нашего приложения. Если вы поступали так же, то просто задайте идентификатор пакета приложения в качестве значения этого ключа;
- `kSecAttrAccount` — здесь указывается ключ, который должен быть удален.

Если вы выполнили все указания, приведенные в разделе 8.2, то на данном этапе связка ключей имеет обобщенный пароль (`kSecClassGenericPassword`) с именем сервиса (`kSecAttrService`), равным идентификатору пакета приложения, а также имеет ключ (`kSecAttrAccount`), равный `Full Name`. Вот что нужно сделать, чтобы удалить этот ключ:

```
#import "AppDelegate.h"
#import <Security/Security.h>

@implementation AppDelegate

- (BOOL) application:(UIApplication *)application
  didFinishLaunchingWithOptions:(NSDictionary *)launchOptions{

    NSString *key = @"Full Name";
    NSString *service = [[NSBundle mainBundle] bundleIdentifier];

    NSDictionary *query = @{
      (__bridge id)kSecClass : (__bridge id)kSecClassGenericPassword,
      (__bridge id)kSecAttrService : service,
      (__bridge id)kSecAttrAccount : key
    };

    OSStatus foundExisting =
    SecItemCopyMatching((__bridge CFDictionaryRef)query, NULL);

    if (foundExisting == errSecSuccess){
      OSStatus deleted = SecItemDelete((__bridge CFDictionaryRef)query);
      if (deleted == errSecSuccess){
        NSLog(@"Successfully deleted the item");
      } else {
        NSLog(@"Failed to delete the item.");
      }
    } else {
      NSLog(@"Did not find the existing value.");
    }

    self.window = [[UIWindow alloc]
                   initWithFrame:[[UIScreen mainScreen] bounds]];
    self.window.backgroundColor = [UIColor whiteColor];
    [self.window makeKeyAndVisible];
    return YES;
}
```

После запуска этой программы (предполагается, что вы выполнили все инструкции из раздела 8.2) вы должны увидеть на консоли `NSLog`, соответствующий успешному удалению. В противном случае вы в любой момент можете считать значение функции `SecItemDelete` и узнать, почему возникла проблема.

## См. также

Раздел 8.2.

# 8.6. Совместное использование данных из связки ключей в нескольких приложениях

## Постановка задачи

Требуется, чтобы хранилищем данных связки ключей могли пользоваться два ваших приложения.

## Решение

Сохраняя данные вашей связки ключей, укажите ключ `kSecAttrAccessGroup` в словаре, передаваемом функции `SecItemAdd`. Значением этого ключа должна быть группа доступа, которую вы найдете в разделе Entitlements (Разрешения) вашего профиля инициализации. Об этом мы говорили во введении к данной главе.

## Обсуждение

Несколько приложений с одного и того же портала разработки могут совместно использовать область связки ключей. Чтобы не усложнять этот пример, в данном разделе рассмотрим взаимодействие всего двух приложений, но описанные методы применимы для любого количества программ.

Чтобы два приложения могли совместно использовать область связки ключей, должны выполняться следующие требования.

1. Оба приложения должны быть подписаны с помощью профиля инициализации, взятого с одного и того же портала для разработки под iOS.
2. Оба приложения должны иметь в своем профиле инициализации один и тот же групповой идентификатор (Group ID). Обычно это идентификатор команды (Team ID), выбираемый Apple. Рекомендую не менять этот групповой идентификатор при создании собственных профилей инициализации.
3. Первое приложение, сохраняющее значение в связке ключей, должно задавать для сохраняемого элемента связки ключей атрибут `kSecAttrAccessGroup`. Эта группа доступа должна совпадать с той, которая указана в вашем профиле инициализации. Прочитайте во введении к этой главе, как извлекать это значение из ваших профилей инициализации.
4. Значение, сохраненное в связке ключей, должно быть сохранено с таким атрибутом `kSecAttrService`, значение которого известно обоим взаимодействующим приложениям. Обычно это идентификатор пакета того приложения, которое сохранило значение в связке ключей. Если вы сами написали оба приложения, то знаете этот идентификатор пакета. Таким образом, в другом вашем приложении можете считать это значение, предоставив идентификатор пакета первого приложения вышеупомянутому ключу.
5. Оба приложения должны обладать идентификацией подписи кода. Это файл настроек `.plist`, содержимое которого в точности совпадает с информацией

из секции Entitlements (Разрешения) вашего профиля инициализации. Затем потребуется указать путь к этому файлу в разделе Code Signing Entitlements (Разрешения для подписи кода) в настройках сборки. Вскоре мы обсудим этот вопрос подробнее.

6. Хотя приложение и подписано профилями инициализации, в которых указаны разрешения (подробнее об этом говорится во введении к данной главе), вам все равно потребуется явно сообщить Xcode об этих разрешениях. Вся информация о разрешениях — это обычный файл `.plist`, содержимое которого очень напоминает тот файл разрешений, вывод которого я продемонстрировал во введении к этой главе.

```
<plist version="1.0">
  <dict>
    <key>application-identifier</key>
    <string>F3FU372W5M.com.pixolity.ios.cookbook.SecondSecurityApp</string>
    <key>com.apple.developer.default-data-protection</key>
    <string>NSFileProtectionComplete</string>
    <key>get-task-allow</key>
    <true/>
    <key>keychain-access-groups</key>
    <array>
      <string>F3FU372W5M.*</string>
    </array>
  </dict>
</plist>
```

Обратите внимание на ключ `keychain-access-groups`. Значение этого ключа указывает группу связки ключей, к которой имеет доступ данное приложение, а именно: `F3FU372W5M.*`. Вам потребуется найти в разделе «Разрешения» свою группу доступа к связке ключей и использовать ее в примерах кода в данном разделе. Мы напишем два приложения. Первое будет заносить в связку ключей информацию, касающуюся группы доступа к связке ключей, а второе будет считывать эту информацию. Приложения будут иметь разные идентификаторы пакетов и, в принципе, будут двумя совершенно разными программами. Однако они смогут совместно использовать определенную область в связке ключей.

Группа доступа является общим идентификатором моей команды и соответствует группе, имеющей доступ к нашей общей связке ключей. Разумеется, в вашем случае это значение будет другим. Воспользуйтесь приемами, изученными в разделе 8.0 этой главы, чтобы извлечь разрешения из ваших профилей инициализации.

Для первого из наших iOS-приложений я задам следующие настройки. Вы должны заменить их собственными.

- *Идентификатор пакета* `com.pixolity.ios.cookbook.SecurityApp`.
- *Группа доступа к связке ключей* `F3FU372W5M.*`.
- *Профиль инициализации*, специально созданный для идентификатора пакета данного приложения.

А вот мои настройки для второго приложения — того, которое будет считывать из связки ключей значения, сохраненные там первым приложением.

- *Идентификатор пакета* `com.pixolity.ios.cookbook.SecurityApp`.
- *Группа доступа к связке ключей* `F3FU372W5M.*`.
- *Профиль инициализации*, специально созданный для идентификатора пакета данного приложения, но отличающийся от аналогичного профиля, созданного для первого приложения.

Самая важная деталь, отличающая первое приложение (сохраняющее связку ключей) от второго (считывающего цепочку ключей), — это идентификаторы пакетов. Первое приложение будет использовать свой идентификатор пакета для сохранения значения в связке ключей, а второе — использовать идентификатор пакета первого приложения для считывания того самого значения из связки ключей. Код очень напоминает тот, что мы написали в разделе 8.2. Разница заключается в том, что новый код будет указывать группу доступа к связке ключей при сохранении данных в этой связке ключей:

```
#import "AppDelegate.h"
#import <Security/Security.h>

@implementation AppDelegate

- (BOOL) application:(UIApplication *)application
  didFinishLaunchingWithOptions:(NSDictionary *)launchOptions{

    NSString *key = @"Full Name";
    NSString *service = [[NSBundle mainBundle] bundleIdentifier];
    NSString *accessGroup = @"F3FU372W5M.*";

    /* Сначала удаляем имеющееся значение, если оно существует. Этого можно
     и не делать, но SecItemAdd завершится с ошибкой, если имеющееся
     значение окажется в связке ключей */
    NSDictionary *queryDictionary = @{
      (__bridge id)kSecClass : (__bridge id)kSecClassGenericPassword,
      (__bridge id)kSecAttrService : service,
      (__bridge id)kSecAttrAccessGroup : accessGroup,
      (__bridge id)kSecAttrAccount : key,
    };

    SecItemDelete((__bridge CFDictionaryRef)queryDictionary);

    /* Затем запишем новое значение в связку ключей */
    NSString *value = @"Steve Jobs";
    NSData *valueData = [value dataUsingEncoding:NSUTF8StringEncoding];

    NSDictionary *secItem = @{
      (__bridge id)kSecClass : (__bridge id)kSecClassGenericPassword,
      (__bridge id)kSecAttrService : service,
      (__bridge id)kSecAttrAccessGroup : accessGroup,
```

```
      (__bridge id)kSecAttrAccount : key,
      (__bridge id)kSecValueData : valueData,
    };

    CFTypeRef result = NULL;
    OSStatus status = SecItemAdd((__bridge CFDictionaryRef)secItem, &result);

    if (status == errSecSuccess){
      NSLog(@"Successfully stored the value");
    } else {
      NSLog(@"Failed to store the value with code: %ld", (long)status);
    }

    self.window = [[UIWindow alloc]
                   initWithFrame:[[UIScreen mainScreen] bounds]];
    self.window.backgroundColor = [UIColor whiteColor];
    [self.window makeKeyAndVisible];
    return YES;
}
```

Код начинается с запрашивания связки ключей для нахождения имеющегося элемента с заданным ключом, именем службы и группой доступа к связке ключей. Если такое значение существует, мы удаляем его из связки ключей. Мы делаем это лишь для того, чтобы позже можно было успешно добавить новое значение. Функция `SecItemAdd` завершится с ошибкой, если вы попытаетесь перезаписать ею имеющееся значение. Поэтому мы удаляем имеющееся значение (если оно существует) и записываем новое. Вы также можете попытаться найти имеющееся значение, обновить его (при наличии такого значения), а если оно отсутствует — записать новое значение. Второй подход более сложен, а поскольку мы выполняем демонстрационное упражнение, обойдемся первым примером.

Однако прежде, чем вы сможете запустить это приложение, необходимо задать в коде разрешения, связанные с подписыванием. Чтобы задать элементы, регламентирующие подписывание кода, выполните следующие шаги.

1. Воспользуйтесь приемами, изученными во введении к этой главе, нужными для извлечения разрешений, находящихся в вашем профиле инициализации.
2. Создайте в вашем приложении новый `.plist`-файл и назовите его `Entitlements.plist`. Вставьте содержимое файла с разрешениями из профиля инициализации — в точности как они есть — в ваш файл `Entitlements.plist` и сохраните.
3. Перейдите в настройки сборки и найдите разрешения подписывания кода (Code Signing Entitlements). Задайте для этого раздела значение `$(TARGET_NAME)/Entitlements.plist`. Оно означает, что Xcode должна найти файл `Entitlements.plist` в целевом каталоге, имеющем такое имя.

Опишу причины, по которым нам требуется это значение. Если создать проект под названием MyProject, Xcode создаст корневой каталог (SCROOT) под названием MyProject. В нем среда создаст еще один подкаталог, который также будет называться MyProject, а уже в этом каталоге будет находиться ваш исходный код. В каталоге SCROOT (то есть в вы-

шестоящем каталоге под названием MyProject) будет создан еще один подкаталог под названием MyProjectTests. В этом подкаталоге будут модульные тесты, интеграционные тесты и тесты пользовательского интерфейса. Под этой структурой вы найдете ваш файл Entitlements.plist (он находится по адресу MyProject/MyProject/Entitlements.plist). Программа поиска разрешений подписывания кода ищет указанный .plist-файл в каталоге SRCROOT, поэтому, если вы предоставите ей значение MyProject/MyProject/Entitlements.plist, это ее вполне устроит! $(TARGET_NAME) в Xcode — это переменная, разрешаемая в имя вашей цели, которое по умолчанию совпадает с именем вашего проекта. Следовательно, в случае с MyProject значение $(TARGET_NAME)/Entitlements.plist будет разрешаться в MyProject/Entitlements.plist.

4. Соберите приложение и убедитесь, что все работает правильно.

Если вы получите сообщение об ошибке примерно следующего содержания:

```
error: The data couldn't be read because it isn't in the correct format[1]
```

это означает, что ваши разрешения записаны в неверном формате. Распространенная ошибка, которую совершают многие iOS-программисты, связана с заполнением файла с разрешениями следующим образом:

```
<plist version="1.0">
  <key>Entitlements</key>
  <dict>
    <key>application-identifier</key>
    <string>F3FU372W5M.com.pixolity.ios.cookbook.SecurityApp</string>
    <key>com.apple.developer.default-data-protection</key>
    <string>NSFileProtectionComplete</string>
    <key>get-task-allow</key>
    <true/>
    <key>keychain-access-groups</key>
    <array>
      <string>F3FU372W5M.*</string>
    </array>
  </dict>
</plist>
```

Обратите внимание: этот файл с разрешениями оформлен неправильно, так как в верхней его части содержится висячий ключ `Entitlements`. Этот ключ потребуется удалить, чтобы ваш файл имел следующий вид:

```
<plist version="1.0">
  <dict>
    <key>application-identifier</key>
    <string>F3FU372W5M.com.pixolity.ios.cookbook.SecurityApp</string>
    <key>com.apple.developer.default-data-protection</key>
    <string>NSFileProtectionComplete</string>
    <key>get-task-allow</key>
    <true/>
    <key>keychain-access-groups</key>
    <array>
```

[1] «Ошибка: данные не могут быть считаны, так как имеют неверный формат». — *Примеч. пер.*

```
      <string>F3FU372W5M.*</string>
    </array>
  </dict>
</plist>
```

Итак, разработка записывающего приложения закончена. Вплотную займемся приложением, которое считывает данные. Это два совершенно самостоятельных подписанных приложения, каждое из которых обладает собственным профилем инициализации:

```
#import "AppDelegate.h"
#import <Security/Security.h>

@implementation AppDelegate

- (BOOL) application:(UIApplication *)application
  didFinishLaunchingWithOptions:(NSDictionary *)launchOptions{

    NSString *key = @"Full Name";
    /* Это не ID пакета того приложения, которое записывало информацию
     в связку ключей. Это и не ID пакета данного приложения. Идентификатор
     пакета данного приложения — com.pixolity.ios.cookbook.SecondSecurityApp */
    NSString *service = @"com.pixolity.ios.cookbook.SecurityApp";
    NSString *accessGroup = @"F3FU372W5M.*";

    NSDictionary *queryDictionary = @{
      (__bridge id)kSecClass : (__bridge id)kSecClassGenericPassword,
      (__bridge id)kSecAttrService : service,
      (__bridge id)kSecAttrAccessGroup : accessGroup,
      (__bridge id)kSecAttrAccount : key,
      (__bridge id)kSecReturnData : (__bridge id)kCFBooleanTrue,
    };

    CFDataRef data = NULL;
    OSStatus found =
    SecItemCopyMatching((__bridge CFDictionaryRef)queryDictionary,
                        (CFTypeRef *)&data);

    if (found == errSecSuccess){
      NSString *value = [[NSString alloc]
                         initWithData:(__bridge_transfer NSData *)data
                         encoding:NSUTF8StringEncoding];

      NSLog(@"Value = %@", value);

    } else {
      NSLog(@"Failed to read the value with error = %ld", (long)found);
    }

    self.window = [[UIWindow alloc]
                   initWithFrame:[[UIScreen mainScreen] bounds]];
```

```
    self.window.backgroundColor = [UIColor whiteColor];
    [self.window makeKeyAndVisible];
    return YES;
}
```

Может потребоваться время, чтобы привыкнуть к тому, как работает связка ключей. Не волнуйтесь, если все не заработает с пол-оборота. Просто внимательно изучите все инструкции, данные в этой главе (особенно в ее введении), чтобы четко понять, что такое группа доступа к связке ключей и как она связана с разрешениями вашего приложения.

## См. также

Разделы 8.0 и 8.2.

# 8.7. Запись и считывание информации связки ключей из iCloud

## Постановка задачи

Требуется сохранить информацию в связке ключей, а также обеспечить хранение этой информации в пользовательской связке ключей, расположенной в облаке iCloud. Так пользователь сможет получать доступ к этой связке с любых имеющихся у него устройств с iOS.

## Решение

При добавлении вашего элемента к связке ключей с помощью функции `SecItemAdd` добавьте ключ `kSecAttrSynchronizable` в словарь, передаваемый этой функции. В качестве значения этого ключа передайте `kCFBooleanTrue`.

## Обсуждение

Когда элементы хранятся в связке ключей с ключом `kSecAttrSynchronizable`, имеющим значение `kCFBooleanTrue`, такие элементы сохраняются и в той пользовательской связке ключей, которая расположена в облаке iCloud. Таким образом, эти элементы будут доступны на всех пользовательских устройствах, если пользователь зашел в них под учетной записью для работы с iCloud. Если вы просто хотите считать значение, которое (как вам точно известно) синхронизировано с пользовательской связкой ключей из iCloud, необходимо указать вышеупомянутый ключ, а также задать для этого ключа значение `kCFBooleanTrue`. В таком случае iOS сможет получить требуемое значение из облака, если еще не сделала этого.

Пример, который мы рассмотрим в этом разделе, на 99 % совпадает с кодом, изученным в разделе 8.6. Разница заключается в следующем: когда мы сохраняем значение в связке ключей или пытаемся его оттуда считать, то указываем в нашем

словаре ключ kSecAttrSynchronizable и задаем для этого ключа значение kCFBooleanTrue. Итак, давайте для начала рассмотрим, как сохранить значение в связке ключей:

```
#import "AppDelegate.h"
#import <Security/Security.h>

@implementation AppDelegate

- (BOOL) application:(UIApplication *)application
  didFinishLaunchingWithOptions:(NSDictionary *)launchOptions{

    NSString *key = @"Full Name";
    NSString *service = [[NSBundle mainBundle] bundleIdentifier];
    NSString *accessGroup = @"F3FU372W5M.*";

    /* Сначала удаляем имеющееся значение, если оно существует. Этого можно
     и не делать, но SecItemAdd завершится с ошибкой, если имеющееся
     значение окажется в связке ключей */
    NSDictionary *queryDictionary = @{
      (__bridge id)kSecClass : (__bridge id)kSecClassGenericPassword,
      (__bridge id)kSecAttrService : service,
      (__bridge id)kSecAttrAccessGroup : accessGroup,
      (__bridge id)kSecAttrAccount : key,
      (__bridge id)kSecAttrSynchronizable : (__bridge id)kCFBooleanTrue
    };

    SecItemDelete((__bridge CFDictionaryRef)queryDictionary);

    /* Затем запишем новое значение в связку ключей */
    NSString *value = @"Steve Jobs";
    NSData *valueData = [value dataUsingEncoding:NSUTF8StringEncoding];

    NSDictionary *secItem = @{
      (__bridge id)kSecClass : (__bridge id)kSecClassGenericPassword,
      (__bridge id)kSecAttrService : service,
      (__bridge id)kSecAttrAccessGroup : accessGroup,
      (__bridge id)kSecAttrAccount : key,
      (__bridge id)kSecValueData : valueData,
      (__bridge id)kSecAttrSynchronizable : (__bridge id)kCFBooleanTrue
    };

    CFTypeRef result = NULL;
    OSStatus status = SecItemAdd((__bridge CFDictionaryRef)secItem, &result);

    if (status == errSecSuccess){
      NSLog(@"Successfully stored the value");
    } else {
      NSLog(@"Failed to store the value with code: %ld", (long)status);
    }

    self.window = [[UIWindow alloc]
```

```
                    initWithFrame:[[UIScreen mainScreen] bounds]];
  self.window.backgroundColor = [UIColor whiteColor];
  [self.window makeKeyAndVisible];
  return YES;
}
```

Пожалуйста, внимательно изучите примечания из раздела 8.6. На данном этапе вы уже должны знать, что группа доступа, упоминаемая во всех приведенных примерах, у каждого разработчика будет называться по-своему. Обычно это командный идентификатор, генерируемый на портале Apple для разработки под iOS для каждого разработчика и обеспечивающий доступ к коду для его команды. Необходимо изменить это значение в ваших примерах и убедиться, что оно совпадает именно с вашим командным идентификатором.

Ранее был приведен код приложения, сохраняющего значения в связке ключей, расположенной в облаке iCloud. Теперь остается написать приложение, считывающее эти данные:

```
#import "AppDelegate.h"
#import <Security/Security.h>

@implementation AppDelegate

- (BOOL) application:(UIApplication *)application
  didFinishLaunchingWithOptions:(NSDictionary *)launchOptions{

    NSString *key = @"Full Name";
    /* Это не ID пакета того приложения, которое записывало информацию
     в связку ключей в iCloud. Это и не ID пакета данного приложения.
       Идентификатор пакета данного приложения - com.pixolity.ios.cookbook.
SecondSecurityApp*/
    NSString *service = @"com.pixolity.ios.cookbook.SecurityApp";
    NSString *accessGroup = @"F3FU372W5M.*";
    NSDictionary *queryDictionary = @{
      (__bridge id)kSecClass : (__bridge id)kSecClassGenericPassword,
      (__bridge id)kSecAttrService : service,
      (__bridge id)kSecAttrAccessGroup : accessGroup,
      (__bridge id)kSecAttrAccount : key,
      (__bridge id)kSecReturnData : (__bridge id)kCFBooleanTrue,
      (__bridge id)kSecAttrSynchronizable : (__bridge id)kCFBooleanTrue
    };
    CFDataRef data = NULL;
    OSStatus found =
    SecItemCopyMatching((__bridge CFDictionaryRef)queryDictionary,
                        (CFTypeRef *)&data);
    if (found == errSecSuccess){
      NSString *value = [[NSString alloc]
                         initWithData:(__bridge_transfer NSData *)data
                         encoding:NSUTF8StringEncoding];

      NSLog(@"Value = %@", value);
```

```
    } else {
      NSLog(@"Failed to read the value with error = %ld", (long)found);
    }

    self.window = [[UIWindow alloc]
                   initWithFrame:[[UIScreen mainScreen] bounds]];
    self.window.backgroundColor = [UIColor whiteColor];
    [self.window makeKeyAndVisible];
    return YES;
}
```

Необходимо отметить еще несколько деталей, связанных с работой со связкой ключей в iCloud.

- В такой связке ключей можно хранить только пароли.
- Связка ключей iCloud работает где угодно — это означает, что она появляется сразу на нескольких устройствах, принадлежащих одному пользователю iCloud. Если вы запишете элемент в связку ключей iCloud, он будет синхронизирован на всех устройствах этого пользователя. Аналогично если вы удалите элемент, то он удалится со всех устройств пользователя, связанных с iCloud. Поэтому в данном случае нужно проявлять особую осторожность.

Также следует отметить, что все остальные приемы, изученные нами в этой главе (например, обновление имеющегося элемента связки ключей — см. раздел 8.4), работают и со связкой ключей, расположенной в iCloud.

## См. также

Разделы 8.0 и 8.6.

# 8.8. Безопасное хранение файлов в песочнице приложения

## Постановка задачи

Требуется, чтобы iOS защищала файлы, расположенные в песочнице вашего приложения, от несанкционированного считывания. Такое считывание, в частности, могут выполнять файловые менеджеры для iOS, которые в изобилии встречаются в Интернете.

## Решение

Сделайте следующие шаги.

1. Выполните все операции, необходимые для создания профиля инициализации приложения, — об этом см. в разделе 8.0. Приложение должно быть подключе-

но к такому идентификатору, для которого действует защита данных (Data Protection).

2. Подпишите ваше приложение профилем инициализации.
3. Задайте разрешения на подписывание кода вашего приложения (следуйте указаниям, изложенным в разделе 8.6).
4. Используйте метод `createFileAtPath:contents:attributes:` экземпляра `NSFileManager`, чтобы сохранить ваш файл. Для свойства `attributes` передайте словарь, содержащий ключ `NSFileProtectionKey`. Этот ключ может иметь одно из следующих значений.
   - `NSFileProtectionNone` — при таком значении сохраненный файл не имеет никакой защиты. Файл, сохраненный с таким уровнем защиты, будет доступен в приложении, сохранившем его на диск, а также любым бесплатным или коммерческим файловым менеджером, взятым из Интернета. Такая программа-менеджер может свободно просматривать файловую систему устройства с iOS, даже если это устройство защищено паролем. Указывая этот ключ, вы получаете возможность как считывать ваш файл, так и записывать в этот файл, в том числе и тогда, когда пользовательское устройство заблокировано.
   - `NSFileProtectionComplete` — это наиболее высокий уровень защиты, который вы можете задать для ваших файлов. При наличии данного ключа приложение сможет считывать такой файл и записывать в него информацию, пока устройство разблокировано. Если устройство заблокировано, считывать файл и записывать в него какие-либо данные невозможно. При применении такого уровня защиты коммерческие файловые менеджеры не могут читать содержимое файла, даже если пользовательское устройство разблокировано.
   - `NSFileProtectionCompleteUnlessOpen` — значение очень похоже на `NSFileProtectionComplete`. Разница понятна уже из названия: вы сможете получить доступ к файлу, если уже открыли его, даже если впоследствии пользователь заблокирует устройство. Итак, открыв файл впервые, вы получаете к нему гарантированный доступ до тех пор, пока существует ваше приложение.
   - `NSFileProtectionCompleteUntilFirstUserAuthentication` — при таком значении ваше приложение сможет считывать файл и заносить в него информацию, как только пользователь в первый раз разблокирует свое устройство. После этого вы можете и далее обращаться к файлу, даже если впоследствии пользователь вновь заблокирует устройство.

   Вот пример:

```
- (NSString *) filePath{
  NSFileManager *fileManager = [[NSFileManager alloc] init];

  NSError *error = nil;
  NSURL *documentFolderUrl = [fileManager URLForDirectory:NSDocumentDirectory
                                                 inDomain:NSUserDomainMask
                                        appropriateForURL:nil
                                                   create:YES
```

```
                                                        error:&error];

  if (error == nil && documentFolderUrl != nil){
    NSString *fileName = @"MyFile.txt";
    NSString *filePath = [documentFolderUrl.path
                          stringByAppendingPathComponent:fileName];
    return filePath;
  }

  return nil;
}

- (BOOL) application:(UIApplication *)application
  didFinishLaunchingWithOptions:(NSDictionary *)launchOptions{

    /*
     Предпосылки:
     1) подписать приложение валидным профилем инициализации;
     2) в вашем профиле должна быть активизирована полная защита файла;
     3) добавить в проект разрешения на подписывание кода.
    */

    NSFileManager *fileManager = [[NSFileManager alloc] init];

    if ([self filePath] != nil){

      NSData *dataToWrite = [@"Hello, World"
                             dataUsingEncoding:NSUTF8StringEncoding];

      NSDictionary *fileAttributes = @{
            NSFileProtectionKey : NSFileProtectionComplete
            };

      BOOL wrote = [fileManager createFileAtPath:[self filePath]
                                        contents:dataToWrite
                                      attributes:fileAttributes];

     if (wrote){
       NSLog(@"Successfully and securely stored the file");
      } else {
        NSLog(@"Failed to write the file");
      }
    }

    self.window = [[UIWindow alloc]
                   initWithFrame:[[UIScreen mainScreen] bounds]];

    self.window.backgroundColor = [UIColor whiteColor];
    [self.window makeKeyAndVisible];
    return YES;
}
```

## Обсуждение

Пользователи доверяют вашим приложениям. Поэтому, когда вы запрашиваете у пользователя определенную личную информацию, например имя и фамилию, пользователь рассчитывает, что эта информация будет храниться в хорошо защищенном месте, где до нее не доберутся хакеры или кто-нибудь, кто получает временный доступ к пользовательскому устройству с iOS.

Предположим, вы пишете приложение для редактирования фотографий. Работая с этим приложением, пользователь может подключить камеру к своему устройству с iOS, импортировать свои фотографии в ваше приложение, а потом пользоваться им для редактирования, сохранения фотографий и раздачи их друзьям. Вы могли бы поступить так, как делают очень многие разработчики приложений: импортировать эти фотографии в папку Documents (Документы), где они сразу будут готовы к редактированию. Но с таким подходом связана одна проблема: любой файловый менеджер для iOS, который можно свободно скачать в Интернете, может считывать содержимое папки Documents (Документы) в любом приложении, даже если устройство заблокировано. Чтобы защитить пользовательские данные, следует активизировать защиту тех файлов, которые вы храните в песочнице приложения. Защита файлов — неотъемлемая часть безопасности пользовательского устройства, в частности, пароля к этому устройству. Допустим, пользователь установил на устройстве пароль, без которого устройство нельзя разблокировать (пусть даже этот пароль совсем простой), и такая блокировка произошла. В таком случае после того, как устройство будет заблокировано, все файлы, сохраненные в песочнице вашего приложения и обладающие ключом `NSFileProtectionComplete`, будут недоступны для посторонних. Прочитать такие файлы не сможет даже файловый менеджер.

Итак, при релизе вашего приложения и даже на этапе его разработки задавайте профили для разработки и распространения программы так, чтобы в них действовала необходимая защита файлов, распространяющаяся на конфиденциальные файлы, находящиеся на диске. Сами определите, какие файлы нуждаются в защите от непрошеных зрителей. Остальные файлы на диске можно и не защищать.

## См. также

Разделы 8.0 и 8.6.

# 8.9. Защита пользовательского интерфейса

## Постановка задачи

Необходимо гарантировать, что пользовательский интерфейс соответствует наиболее распространенным правилам безопасности, действующим в iOS.

## Решение

Следуйте приведенным далее указаниям.

- Обеспечьте, чтобы вся информация, вводимая пользователем в поля паролей и другие защищенные поля, попадала в экземпляры `UITextField`, чьим свойствам `secureTextEntry` присвоено значение `YES`.
- Если пользователь находится на экране, содержащем персональную информацию, например номер кредитной карточки или домашний адрес, присвойте свойству `hidden` главного окна вашего приложения значение `YES` (в методе `applicationWillResignActive:` делегата вашего приложения). Чтобы само окно отображалось на экране, тому же самому свойству нужно присвоить значение `NO` в методе `applicationDidBecomeActive:` делегата приложения. Так вы гарантируете, что на скриншоте пользовательского интерфейса (iOS снимает такой скриншот, когда приложение переходит в фоновый режим) не будет отражаться никакое содержимое вашего окна. Apple рекомендует действовать именно так.
- Обязательно валидируйте пользовательский ввод в текстовых полях/видах перед отправкой этой информации на сервер.
- Пользуясь механизмами, изученными в этой главе, защищайте пользовательские записи, если храните их в файлах на диске или в связке ключей.
- На тех экранах, где вы принимаете пароль или числовой код для аутентификации, очищайте такие поля с кодом/паролем, как только контроллер вида перестает отображаться на экране. Если вы не будете уступать и высвобождать эти контроллеры видов, их содержимое будет сохраняться в памяти. В частности, в памяти будут находиться записи из защищенных текстовых полей, находящихся в этих контроллерах видов. Целесообразно высвобождать память, содержащую конфиденциальные данные, сразу же после того, как исчезает необходимость в этих данных.

## Обсуждение

В этом списке лишь второй элемент требует дополнительного объяснения. Когда пользователь видит окно приложения на экране своего устройства с iOS и переводит это приложение в фоновый режим, возвращаясь на главный экран (нажимая Home), iOS помещает приложение в неактивное состояние. Когда приложение в таком состоянии переведено в фоновый режим, iOS делает скриншот пользовательского интерфейса этого приложения (в точном соответствии с изображением на экране) и сохраняет этот файл в каталоге Library/Caches/Snapshots/. Данный каталог находится в песочнице вашего приложения. Как только пользователь вновь переводит приложение в приоритетный режим, iOS сразу же отображает этот скриншот, и он остается на экране до тех пор, пока приложение не «оживет» окончательно и не примет управление экраном. Поэтому переход из фонового в приоритетный режим в iOS получается очень плавным. Но хотя такая практика и очень положительна с точки зрения удобства использования (UX), она привносит определенную проблему в области безопасности. Дело в том, что если на скриншоте была зафиксирована конфиденциальная информация, то она будет сохранена и на диске.

Мы не можем полностью отключить эту функцию в iOS, однако можем нейтрализовать ее негативное влияние на безопасность приложения. Чтобы это сделать (кстати, такая практика рекомендуется Apple), нужно накрыть основное окно нашего приложения другим видом либо скрыть содержимое этого окна, устанавливая свойство `hidden` окна приложения в значение `YES`, когда приложение становится неактивным. При переходе приложения в активное состояние мы вновь присваиваем этому свойству значение `NO` (и окно снова становится видимым).

iOS-разработчики, стремящиеся выполнять это требование безопасности, часто совершают одну ошибку. Они пытаются устанавливать в YES или NO значение свойства hidden экземпляра keyWindow. Даже хотя окно keyWindow экземпляра приложения будет валидным окном в момент, когда приложение становится неактивным, в момент «оживления» приложения keyWindow равно nil — то есть указывает в никуда. Следовательно, во избежание возможных ошибок просто пользуйтесь свойством window делегата приложения, отображая или скрывая окно.

Другая проблема, касающаяся безопасности приложения, связана с «оседанием» персональных данных в контроллерах видов. Предположим, у вас есть контроллер вида для входа в систему. Здесь пользователь вводит свои имя и пароль. Как только будет нажата кнопка, которая зачастую называется Login (Вход в систему), вы отправляете учетные данные пользователя на сервер по сетевому HTTPS-соединению. Когда произойдет аутентификация пользователя, вы выдвигаете на экран другой контроллер вида. Проблема, связанная с таким подходом, заключается в том, что имя и пароль, введенные пользователем на предыдущем экране, по-прежнему остаются в памяти (как и сам контроллер вида). Как вы помните, навигационный контроллер включает целый стек контроллеров видов.

Чтобы справиться с этой проблемой и повысить безопасность пользовательского интерфейса, можно присвоить свойству `text` защищенных текстовых полей значение `nil`. Это делается в тот самый момент, когда на экран выдвигается второй контроллер вида. Другой способ — переопределить метод экземпляра `viewWillDisappear:` контроллера вида для входа в систему, а также установить свойство `text` текстовых полей в `nil` прямо здесь. Тем не менее такой подход требует известной осторожности, так как упомянутый метод экземпляра контроллера вида вызывается *всякий раз*, когда контроллер исчезает с экрана — например, когда пользователь покидает вкладку с контроллером вида, уходит на другую вкладку, а потом возвращается на первую. Действительно, при таком переходе контроллер вида успевает и исчезнуть с экрана, и вновь там появиться. Поэтому если пользователь всего лишь перешел на соседнюю вкладку, а вы уже стерли всю информацию, введенную в поля, то при возвращении на первую вкладку пользователь найдет там лишь пустые окошки и будет вынужден заносить всю информацию заново. Разработка всегда ведется в соответствии с поставленными бизнес-требованиями, поэтому не существует однозначного ответа на вопрос о том, как справиться с такой ситуацией.

## См. также

Разделы 8.2 и 8.8.

# 9 Core Location и карты

## 9.0. Введение

Фреймворки Core Location и Map Kit можно применять для создания приложений, приспособленных для обработки геолокационной информации (информации о местоположении) и картографических приложений. Фреймворк Core Location использует оборудование устройства для определения актуального местонахождения этого устройства. Фреймворк Map Kit, в свою очередь, позволяет программе отображать для пользователя карты, снабжать карту определенными аннотациями и т. д. С чисто программистской точки зрения доступность геолокационных сервисов зависит от наличия на устройстве необходимого оборудования; если оборудование имеется, то оно должно быть активизировано и подключено для работы с фреймворком Core Location или Map Kit. Устройство с операционной системой iOS, оснащенное службами GPS (системы глобального позиционирования), позволяет работать с технологиями 2G, EDGE, 3G, 4G и другими, которые помогают определять местоположение пользователя. В настоящее время практически на любых устройствах с iOS поддерживаются геолокационные службы, но программисту рекомендуется проверять доступность таких сервисов и приступать к работе с ними, лишь убедившись в их наличии. Ведь мы и в самом деле не можем знать наверняка, не будет ли в будущем Apple выпускать какое-либо устройство, на котором не будет всего оборудования, необходимого для обеспечения геолокационных функций.

В новом компиляторе LLVM, предоставляемом в Xcode для iOS 7, Apple реализовала концепцию модулей. В более ранних версиях SDK и Xcode для использования фреймворков Core Location и Map Kit требовалось вручную импортировать эти фреймворки в целевой проект. Но с появлением модулей для добавления этих фреймворков требуется всего лишь импортировать их заголовочные файлы в классы проекта, вот так:

```
#import <CoreLocation/CoreLocation.h>
#import <MapKit/MapKit.h>
```

И все. Фреймворки Core Location и Map Kit окажутся в ваших проектах.

# 9.1. Создание картографического вида

## Постановка задачи

Необходимо инстанцировать и отобразить карту в экранном виде.

## Решение

Создайте экземпляр класса `MKMapView`, после чего добавьте его к виду либо присвойте подвиду контроллера вашего вида. Вот пример `.h`-файла такого контроллера вида, в котором создается экземпляр `MKMapView`, после чего этот вид отображается в полноэкранном режиме:

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

@interface ViewController ()
@property (nonatomic, strong) MKMapView *myMapView;
@end

@implementation ViewController
```

Это обычный корневой контроллер вида, содержащий переменную `MKMapView`. В следующем коде в реализации данного контроллера вида (`.m`-файле) мы инициализируем карту и зададим для нее тип `Satellite`:

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

  self.view.backgroundColor = [UIColor whiteColor];

  self.myMapView = [[MKMapView alloc]
                    initWithFrame:self.view.bounds];
    /* Задаем Satellite в качестве типа карты. */
    self.myMapView.mapType = MKMapTypeSatellite;
    self.myMapView.autoresizingMask =
    UIViewAutoresizingFlexibleWidth |
    UIViewAutoresizingFlexibleHeight;

  /* Добавляем карту к нашему виду. */
  [self.view addSubview:self.myMapView];

}
```

## Обсуждение

Создать экземпляр класса `MKMapView` довольно легко. Можно просто присвоить ему рамку, воспользовавшись его же конструктором, а после того как карта будет создана,

добавить ее в качестве подвида к виду, который в настоящий момент отображается на экране. И все, мы сможем просматривать карту.

MKMapView — это подкласс UIView. Таким образом, можно манипулировать любым картографическим видом тем же способом, каким вы работаете с экземпляром UIView. К примеру, мы пользуемся свойством UIView для того, чтобы вставить в вид его свойство backgroundColor.

Вы, наверное, уже заметили, что у класса `MKMapView` есть свойство под названием `mapType`, характеризующее тип карты. Карта может быть спутниковой, стандартной или гибридной. В примере мы пользуемся картой спутникового типа (рис. 9.1).


**Рис. 9.1.** Вид карты, выполненной со спутника

Можно изменить визуальное представление карты определенного типа, воспользовавшись свойством `mapType` экземпляра `MKMapView`. Это свойство может принимать следующие значения:

- MKMapTypeStandard — применяется для отображения стандартной карты (задается по умолчанию);
- MKMapTypeSatellite — позволяет отобразить вид карты, выполненной со спутника (как показано на рис. 9.1);
- MKMapTypeHybrid — дает возможность накладывать стандартную карту на спутниковую.

# 9.2. Обработка событий картографического вида

## Постановка задачи

Необходимо обрабатывать различные события, которые картографический вид может посылать своему делегату.

## Решение

Присвойте объект делегата, соответствующий протоколу MKMapViewDelegate, свойству delegate, которое относится к экземпляру класса MKMapView:

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

  /* Создаем карту размером с наш вид. */
  self.myMapView = [[MKMapView alloc]
                    initWithFrame:self.view.bounds];

  /* Задаем Satellite в качестве типа карты. */
  self.myMapView.mapType = MKMapTypeSatellite;

  self.myMapView.delegate = self;

  self.myMapView.autoresizingMask =
  UIViewAutoresizingFlexibleWidth |
  UIViewAutoresizingFlexibleHeight;

  /* Добавляем карту к нашему виду. */
  [self.view addSubview:self.myMapView];
}
```

Этот код легко запустить в методе viewDidLoad, относящемся к объекту контроллера вида, если объект имеет свойство MapView типа MKMapView:

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

@interface ViewController () <MKMapViewDelegate>
```

```
@property (nonatomic, strong) MKMapView *myMapView;
@end
@implementation ViewController
```

## Обсуждение

Объект, являющийся делегатом экземпляра класса `MKMapView`, должен реализовывать методы, описанные в протоколе `MKMapViewDelegate`. Эти методы необходимы для получения различных сообщений от картографического вида и, как будет показано позднее, для предоставления информации картографическому виду. В протоколе `MKMapViewDelegate` определяются различные методы, в том числе метод `mapViewWillStartLoadingMap:`, вызываемый в объекте делегата всякий раз, когда начинается процесс загрузки карты. Не забывайте, что делегат для картографического вида не является обязательным объектом, то есть картографические виды можно создавать и не присваивая им делегатов. Просто картографические виды, лишенные делегатов, не будут реагировать на действия пользователя.

Вот список некоторых методов, объявляемых в протоколе `MKMapViewDelegate` (здесь также рассказано, о чем они должны сообщать объекту-делегату экземпляра `MKMapView`):

- `mapViewWillStartLoadingMap:` — вызывается применительно к объекту делегата всякий раз, когда картографический вид начинает загружать данные, обеспечивающие визуальное представление карты пользователю;
- `mapView:viewForAnnotation:` — вызывается применительно к объекту делегата всякий раз, когда картографический вид требует от экземпляра `MKAnnotationView` снабдить карту визуальными аннотациями. Подробнее об этом механизме будет рассказано в разделе 9.4;
- `mapViewWillStartLocatingUser:` — как понятно из названия, метод вызывается применительно к объекту делегата всякий раз, когда картографический вид приступает к определению местоположения пользователя. Подробнее о том, как сделать это, будет рассказано в разделе 9.3;
- `mapView:regionDidChangeAnimated:` — вызывается применительно к объекту делегата всякий раз, когда изменяется регион, отображаемый на карте.

## См. также

Разделы 9.3 и 9.4.

# 9.3. Отметка местоположения устройства

## Постановка задачи

Необходимо найти широту и долготу той точки, в которой находится устройство.

## Решение

Воспользуйтесь классом CLLocationManager:

```
- (void)viewDidLoad {
  [super viewDidLoad];
  if ([CLLocationManager locationServicesEnabled]){
    self.myLocationManager = [[CLLocationManager alloc] init];
    self.myLocationManager.delegate = self;

    [self.myLocationManager startUpdatingLocation];
  } else {
    /* Геолокационные службы не активизированы.
       Попробуйте исправить ситуацию: например предложите пользователю
       включить геолокационные службы. */
    NSLog(@"Location services are not enabled");
  }
}
```

В данном коде myLocationManager — это свойство типа CLLocationManager. В приведенном примере кода данный класс также является делегатом диспетчера местоположения (Location Manager).

## Обсуждение

Фреймворк Core Location, входящий в состав комплекта SDK, предоставляет программисту функционал, который позволяет определять актуальное положение устройства с системой iOS в пространстве. Поскольку в iOS пользователь может отключать определение местоположения в разделе Settings (Настройки), то мы перед тем, как инстанцировать объект типа CLLocationManager, проверим, работают ли на устройстве геолокационные службы.

Объект, являющийся делегатом CLLocationManager, должен соответствовать протоколу CLLocationManagerDelegate.

Вот как мы объявим объект нашего диспетчера местоположения в .h-файле контроллера вида (создавать экземпляр CLLocationManager может и объект, не являющийся контроллером вида):

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

@interface ViewController () <CLLocationManagerDelegate>
@property (nonatomic, strong) CLLocationManager *myLocationManager;
@end

@implementation ViewController
```

Контроллер нашего вида будет иметь следующую реализацию:

```
- (void)locationManager:(CLLocationManager *)manager
    didUpdateToLocation:(CLLocation *)newLocation
           fromLocation:(CLLocation *)oldLocation{

  /* Получена информация о новом местоположении. */

  NSLog(@"Latitude = %f", newLocation.coordinate.latitude);
  NSLog(@"Longitude = %f", newLocation.coordinate.longitude);

}

- (void)locationManager:(CLLocationManager *)manager
      didFailWithError:(NSError *)error{

  /* Не удалось получить информацию о местоположении пользователя. */

}

- (void)viewDidLoad {
  [super viewDidLoad];

  if ([CLLocationManager locationServicesEnabled]){
    self.myLocationManager = [[CLLocationManager alloc] init];
    self.myLocationManager.delegate = self;


    [self.myLocationManager startUpdatingLocation];
  } else {
    /* Геолокационные службы не активизированы.
       Попробуйте исправить ситуацию: например предложите пользователю
       включить геолокационные службы. */
  NSLog(@"Location services are not enabled");
  }

}
```

Метод экземпляра `startUpdateLocation`, относящийся к классу `CLLocationManager`, сообщает делегату о том, удалось или нет получить информацию о местоположении пользователя. Это делается с помощью методов `location Manager:didUpdate-ToLocation:fromLocation:` и `locationManager:didFailWithError:` объекта делегата, именно в таком порядке.

# 9.4. Отображение маркеров в картографическом виде

## Постановка задачи

Необходимо указать пользователю конкретное место на карте.

## Решение

Воспользуйтесь встроенными аннотациями для картографических видов. Для этого выполните следующие шаги.

1. Создайте новый класс и назовите его `MyAnnotation`.
2. Убедитесь, что этот класс соответствует протоколу `MKAnnotation`.
3. Определите свойство типа `CLLocationCoordinate2D` для этого класса и назовите данное свойство `coordinate`. Убедитесь, что задали это свойство как `readonly` (только для чтения), поскольку свойство `coordinate` в соответствии с протоколом `MKAnnotation` определяется как `readonly`.
4. Далее можно (но не обязательно) определить два свойства типа `NSString`, а именно `title` и `subtitle`, которые могут содержать заголовок и подзаголовок вашего аннотирующего вида. Оба этих свойства также будут `readonly`.
5. Создайте для вашего класса метод-инициализатор. Этот метод будет принимать параметр типа `CLLocationCoordinate2D`. В этом методе присвойте переданный параметр местоположения тому свойству, которое мы определили на этапе 3. Поскольку это свойство является `readonly`, его невозможно присвоить с помощью кода вне области видимости данного класса. Следовательно, инициализатор этого класса действует здесь как перемычка и позволяет опосредованно присваивать значение этому свойству. Такие же операции мы осуществим со свойствами `title` и `subtitle`.
6. Инстанцируйте класс `MyAnnotation` и добавьте его к вашей карте с помощью метода `addAnnotation:`, относящегося к классу `MKMapView`.

## Обсуждение

Как было рассказано в подразделе «Решение» данного раздела, нам следует создать объект, соответствующий протоколу `MKAnnotation`, а позже инстанцировать этот объект и передать ему карту для отображения. `.h`-файл этого объекта будет записываться так:

```
#import <Foundation/Foundation.h>
#import <MapKit/MapKit.h>

@interface MyAnnotation : NSObject <MKAnnotation>

@property (nonatomic, readonly) CLLocationCoordinate2D coordinate;
@property (nonatomic, copy, readonly) NSString *title;
@property (nonatomic, copy, readonly) NSString *subtitle;

- (instancetype)initWithCoordinates:(CLLocationCoordinate2D)paramCoordinates
                              title:(NSString *)paramTitle
                           subTitle:(NSString *)paramSubTitle;
@end
```

В `.m`-файле класса `MyAnnotation` мы создаем класс, отвечающий за отображение геолокационной информации, и делаем это следующим образом:

```
#import "MyAnnotation.h"

@implementation MyAnnotation

- (instancetype)initWithCoordinates:(CLLocationCoordinate2D)paramCoordinates
                         title:(NSString *)paramTitle
                      subTitle:(NSString *)paramSubTitle{

  self = [super init];

  if (self != nil){
    coordinate = paramCoordinates;
    title = paramTitle;
    subtitle = paramSubTitle;
  }

  return(self);

}

@end
```

Позже мы инстанцируем этот класс и добавим его к нашей карте, например к .m-файлу того контроллера вида, который создает и отображает картографический вид:

```
#import "ViewController.h"
#import "MyAnnotation.h"
#import <MapKit/MapKit.h>

@interface ViewController () <MKMapViewDelegate>
@property (nonatomic, strong) MKMapView *myMapView;
@end

@implementation ViewController

- (void)viewDidLoad {
  [super viewDidLoad];

  /* Создаем карту такого же размера, как и наш вид. */
  self.myMapView = [[MKMapView alloc]
                    initWithFrame:self.view.bounds];

  self.myMapView.delegate = self;

  /* Задаем для карты тип Standard. */
  self.myMapView.mapType = MKMapTypeStandard;
  self.myMapView.autoresizingMask =
    UIViewAutoresizingFlexibleWidth |
    UIViewAutoresizingFlexibleHeight;

  /* Добавляем ее к нашему виду. */
```

```
    [self.view addSubview:self.myMapView];

    /* Это просто один образец местоположения. */
    CLLocationCoordinate2D location =
      CLLocationCoordinate2DMake(50.82191692907181, -0.13811767101287842);

    /* Создаем аннотацию, используя информацию о местоположении. */
    MyAnnotation *annotation =
    [[MyAnnotation alloc] initWithCoordinates:location
                                        title:@"My Title"
                                     subTitle:@"My Sub Title"];

    /* И наконец, добавляем аннотацию на карту. */
    [self.myMapView addAnnotation:annotation];

@end
```

На рис. 9.2 показан вывод данной программы в симуляторе iPhone.

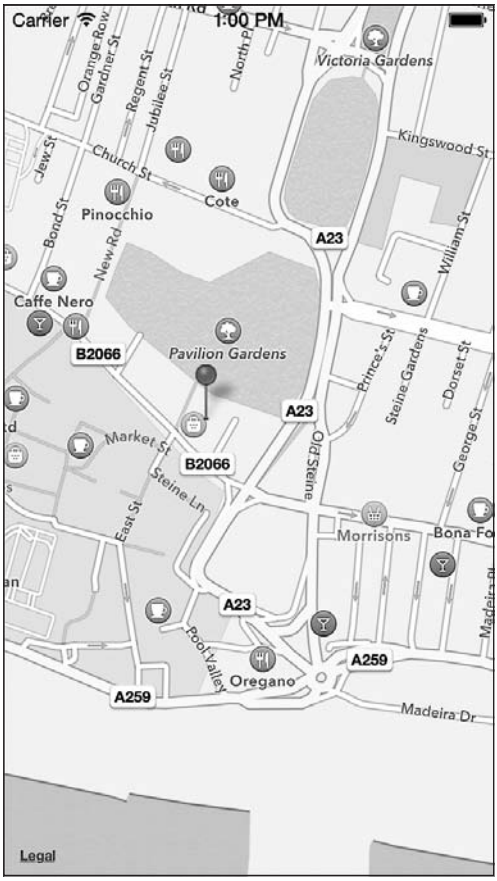


**Рис. 9.2.** Интегрированный в систему стандартный маркер, отображенный на карте

# 9.5. Отображение разноцветных маркеров в картографическом виде

## Постановка задачи

По умолчанию маркеры-индикаторы, которыми отмечаются точки на карте, — красного цвета. Необходимо отображать маркеры различных цветов, а не только стандартного красного.

## Решение

Возвращайте вашему картографическому виду экземпляры `MKPinAnnotationView`. Это делается с помощью метода делегата `mapView:viewForAnnotation:`.

Каждая аннотация, добавляемая к экземпляру `MKMapView`, соответствует конкретному виду, который отображается поверх картографического вида. Такие всплывающие виды называются *аннотирующими* (Annotation Views).

Аннотирующий вид — это объект типа `MKAnnotationView`, он является подклассом от `UIView`. Если объект делегата картографического вида реализует метод делегата `mapView:viewForAnnotation:`, то объект делегата должен будет возвращать экземпляры класса `MKAnnotationView`, чтобы отображать (а при необходимости — настраивать) аннотирующие виды, которые выводятся поверх картографического вида.

## Обсуждение

Чтобы обеспечить в нашей программе возможность настройки цвета меток (цвет будем выбирать из стандартной палитры, предусмотренной для меток в SDK), которые ставятся на картографическом виде для представления аннотаций, нам понадобится возвращать в методе делегата `mapView:viewForAnnotation:` не экземпляр класса `MKAnnotationView`, а экземпляр класса `MKPinAnnotationView`. Не забывайте, что класс `MKPinAnnotationView` является подклассом `MKAnnotationView`.

```
- (MKAnnotationView *)mapView:(MKMapView *)mapView
            viewForAnnotation:(id <MKAnnotation>)annotation{

  MKAnnotationView *result = nil;

  if ([annotation isKindOfClass:[MyAnnotation class]] == NO){
    return result;
  }

  if ([mapView isEqual:self.myMapView] == NO){
    /* Мы собираемся обработать это событие только для того Map View,
       который создали ранее. */
    return result;
  }
```

```
  /* Сначала приводим тип той аннотации, для которой этот Map View
     запустил данное сообщение делегата. */
  MyAnnotation *senderAnnotation = (MyAnnotation *)annotation;

  /* С помощью метода класса, определенного нами в собственном
     классе аннотаций, мы попытаемся сделать многоразовый идентификатор
     для того маркера, который сейчас создаем. */
  NSString *pinReusableIdentifier =
  [MyAnnotation
    reusableIdentifierforPinColor:senderAnnotation.pinColor];

  /* Пользуясь идентификатором, полученным ранее, попытаемся
     повторно применить маркер в отправляющем Map View. */
  MKPinAnnotationView *annotationView = (MKPinAnnotationView *)
  [mapView
  dequeueReusableAnnotationViewWithIdentifier:pinReusableIdentifier];

  if (annotationView == nil){
    /* Если нам не удастся повторно использовать имеющийся маркер,
       создадим новый. */
    annotationView =  [[MKPinAnnotationView alloc]
                       initWithAnnotation:senderAnnotation
                       reuseIdentifier:pinReusableIdentifier];

    /* Убеждаемся, что видны выноски поверх каждого маркера в случае,
       если мы присвоили каждому маркеру заголовок и/или подзаголовок. */
    [annotationView setCanShowCallout:YES];
  }

  /* Теперь (независимо от того, использовали мы многоразовый маркер
     или создали новый) убеждаемся, что цвет маркера совпадает с цветом
     аннотации. */
  annotationView.pinColor = senderAnnotation.pinColor;

  result = annotationView;

  return result;
}
```

При многократном использовании аннотирующего вида ему присваивается идентификатор (строка `NSString`). Определяя, маркер какого типа вы хотели бы отобразить на карте, и задавая уникальный идентификатор для маркера каждого типа (например, к одному типу могут относиться красные маркеры, а к другому — синие), следует многократно использовать маркеры нужного типа, применяя метод экземпляра `dequeueReusableAnnotationViewWithIdentifier:`, относящийся к классу `MKMapView`. Это показано в следующем коде.

Мы запрограммировали механизм получения уникальных идентификаторов каждого маркера в собственном классе `MyAnnotation`. Вот `.h`-файл класса `MyAnnotation`:

```
#import <Foundation/Foundation.h>
#import <MapKit/MapKit.h>

/* Это стандартные цвета меток, присутствующие в SDK. Мы задаем уникальные
   идентификаторы для каждого маркера в соответствии с его цветом, чтобы
   позже можно было снова использовать созданные ранее маркеры в связи
   с тем же цветом, для которого они создавались. */

extern NSString *const kReusablePinRed;
extern NSString *const kReusablePinGreen;
extern NSString *const kReusablePinPurple;
@interface MyAnnotation : NSObject <MKAnnotation>

/* unsafe_unretained, так как это не объект. Этот шаг можно пропустить
   и оставить принятие этого решения компилятору. weak или strong
   не сработают, так как это не объект. */
@property (nonatomic, unsafe_unretained, readonly)
  CLLocationCoordinate2D coordinate;

@property (nonatomic, copy) NSString *title;
@property (nonatomic, copy) NSString *subtitle;

/* unsafe_unretained по той же причине, что и для свойства coordinate */
@property (nonatomic, unsafe_unretained) MKPinAnnotationColor pinColor;

- (instancetype)initWithCoordinates:(CLLocationCoordinate2D)paramCoordinates
                      title:(NSString*)paramTitle
                   subTitle:(NSString*)paramSubTitle;

+ (NSString *)   reusableIdentifierforPinColor
                 :(MKPinAnnotationColor)paramColor;
@end
```

Аннотация не то же самое, что аннотирующий вид. Аннотация — это место, которое вы хотите указать на карте, а аннотирующий вид — это визуальное представление, в котором эта аннотация всплывает над картой (то есть вид). Класс `MyAnnotation` соответствует аннотации, а не аннотирующему виду. Когда мы создаем аннотацию путем инстанцирования класса `MyAnnotation`, мы можем присвоить ей цвет, задействовав определенное и реализованное нами же свойство `pinColor`. Когда картографический вид должен будет отобразить аннотацию, картографический вид вызовет метод делегата `mapView:viewForAnnotation:` и запросит у этого делегата аннотирующий вид. В параметре `forAnnotation` данного метода сообщается аннотация, которую необходимо отобразить. Получая ссылку на аннотацию, мы можем привести тип аннотации к экземпляру `MyAnnotation`, получить ее свойство `pinColor` и, основываясь на этих данных, создать экземпляр класса `MKPinAnnotationView`. У этого экземпляра будет информация о заданном цвете маркера, которую мы вернем картографическому виду.

Вот .m-файл MyAnnotation:

```
#import "MyAnnotation.h"

NSString *const kReusablePinRed = @"Red";
NSString *const kReusablePinGreen = @"Green";
NSString *const kReusablePinPurple = @"Purple";

@implementation MyAnnotation
+ (NSString *)   reusableIdentifierforPinColor
                 :(MKPinAnnotationColor)paramColor{

  NSString *result = nil;

  switch (paramColor){
    case MKPinAnnotationColorRed:{
      result = REUSABLE_PIN_RED;
      break;
    }
    case MKPinAnnotationColorGreen:{
      result = REUSABLE_PIN_GREEN;
      break;
    }
    case MKPinAnnotationColorPurple:{
      result = REUSABLE_PIN_PURPLE;
      break;
    }
  }

  return result;
}

- (instancetype)initWithCoordinates:(CLLocationCoordinate2D)paramCoordinates
                      title:(NSString*)paramTitle
                   subTitle:(NSString*)paramSubTitle{

  self = [super init];

  if (self != nil){
    _coordinate = paramCoordinates;
    _title = paramTitle;
    _subtitle = paramSubTitle;
    _pinColor = MKPinAnnotationColorGreen;
  }

  return self;
}

@end
```

Выполнив реализацию класса MyAnnotation, его нужно задействовать в приложении (в данном примере мы воспользуемся контроллером вида). Вот верхняя часть файла реализации контроллера вида:

```
#import "ViewController.h"
#import "MyAnnotation.h"
#import <MapKit/MapKit.h>

@interface ViewController () <MKMapViewDelegate>
@property (nonatomic, strong) MKMapView *myMapView;
@end
@implementation ViewControllerРеализация в файле .m будет такой:

- (MKAnnotationView *)mapView:(MKMapView *)mapView
            viewForAnnotation:(id <MKAnnotation>)annotation{

  MKAnnotationView *result = nil;

  if ([annotation isKindOfClass:[MyAnnotation class]] == NO){
    return result;
  }

  if ([mapView isEqual:self.myMapView] == NO){
    /* Мы собираемся обработать это событие только для того Map View,
       который мы создали ранее. */
    return result;
  }

  /* Сначала приводим тип той аннотации, для которой этот Map View
     запустил данное сообщение делегата. */
  MyAnnotation *senderAnnotation = (MyAnnotation *)annotation;

  /* С помощью метода класса, определенного в нашем собственном
     классе аннотаций, попытаемся сделать многоразовый идентификатор
     для того маркера, который сейчас создаем. */
  NSString *pinReusableIdentifier =
  [MyAnnotation
    reusableIdentifierforPinColor:senderAnnotation.pinColor];

  /* Пользуясь идентификатором, полученным ранее, попытаемся
     повторно применить маркер в отправляющем Map View. */
  MKPinAnnotationView *annotationView = (MKPinAnnotationView *)
  [mapView
    dequeueReusableAnnotationViewWithIdentifier:pinReusableIdentifier];

  if (annotationView == nil){
    /* Если нам не удастся повторно использовать имеющийся маркер,
       создадим новый. */
    annotationView =  [[MKPinAnnotationView alloc]
                       initWithAnnotation:senderAnnotation
                       reuseIdentifier:pinReusableIdentifier];
```

```
    /* Убеждаемся, что видны выноски поверх каждого маркера в случае,
       если мы присвоили каждому маркеру заголовок и/или подзаголовок. */
    [annotationView setCanShowCallout:YES];
  }

  /* Теперь (независимо от того, использовали мы многоразовый маркер
     или создали новый) убеждаемся, что цвет маркера совпадает с цветом
     аннотации. */
  annotationView.pinColor = senderAnnotation.pinColor;

  result = annotationView;

  return result;
}

- (void)viewDidLoad {
  [super viewDidLoad];

  /* Создаем карту такого же размера, как и наш вид. */
  self.myMapView = [[MKMapView alloc]
                    initWithFrame:self.view.bounds];

  self.myMapView.delegate = self;

  /* Задаем для карты тип Standard. */
  self.myMapView.mapType = MKMapTypeStandard;

  self.myMapView.autoresizingMask =
    UIViewAutoresizingFlexibleWidth |
    UIViewAutoresizingFlexibleHeight;

  /* Добавляем ее к нашему виду. */
  [self.view addSubview:self.myMapView];

  /* Это просто один образец местоположения. */
  CLLocationCoordinate2D location;
  location.latitude = 50.82191692907181;
  location.longitude = -0.13811767101287842;

  /* Создаем аннотацию, используя информацию о местоположении. */
  MyAnnotation *annotation =
  [[MyAnnotation alloc] initWithCoordinates:location
                                      title:@"My Title"
                                   subTitle:@"My Sub Title"];

  annotation.pinColor = MKPinAnnotationColorPurple;

  /* И наконец, добавляем аннотацию на карту. */
  [self.myMapView addAnnotation:annotation];

}
```

Результат проделанной работы показан на рис. 9.3.

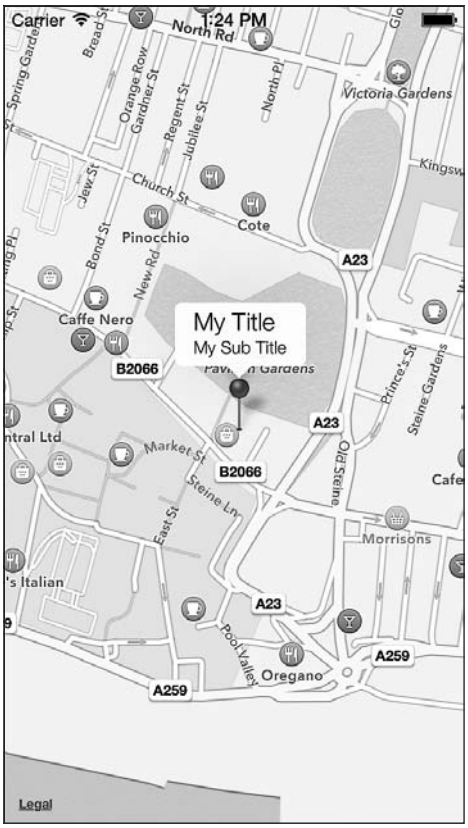


**Рис. 9.3.** Маркер альтернативного цвета, отображенный в картографическом виде

# 9.6. Отображение пользовательских маркеров в картографическом виде

## Постановка задачи

Вместо стандартных маркеров, присутствующих в iOS SDK, требуется использовать на карте в таком качестве наши собственные изображения.

## Решение

Загружаем произвольное изображение в экземпляр класса `UIImage` и присваиваем этот экземпляр свойству `image` экземпляра `MKAnnotationView`. В результате выбранное нами изображение возвращается карте в виде маркера:

```
- (MKAnnotationView *)mapView:(MKMapView *)mapView
            viewForAnnotation:(id <MKAnnotation>)annotation{
```

```
  MKAnnotationView *result = nil;

  if ([annotation isKindOfClass:[MyAnnotation class]] == NO){
    return result;
  }
}

  if ([mapView isEqual:self.myMapView] == NO){
    /* Мы собираемся обработать это событие только для того Map View,
       который создали ранее. */
    return result;
  }

   /* Сначала приводим тип той аннотации, для которой этот Map View
      запустил данное сообщение делегата. */
  MyAnnotation *senderAnnotation = (MyAnnotation *)annotation;

  /* С помощью метода класса, определенного в нашем собственном
     классе аннотаций, попытаемся сделать многоразовый идентификатор
     для того маркера, который сейчас создаем. */
  NSString *pinReusableIdentifier =
  [MyAnnotation
    reusableIdentifierforPinColor:senderAnnotation.pinColor];

  /* Пользуясь идентификатором, полученным ранее, попытаемся повторно
     применить маркер в отправляющем Map View. */
  MKPinAnnotationView *annotationView = (MKPinAnnotationView *)
  [mapView
    dequeueReusableAnnotationViewWithIdentifier:
    pinReusableIdentifier];

  if (annotationView == nil){
    /* Если нам не удастся повторно использовать имеющийся маркер,
       создадим новый. */
    annotationView =
    [[MKPinAnnotationView alloc]  initWithAnnotation:senderAnnotation
                                     reuseIdentifier:pinReusableIdentifier];

    /* Убеждаемся, что видны выноски поверх каждого маркера в случае,
       если мы присвоили каждому маркеру заголовок и/или подзаголовок. */
    annotationView.canShowCallout = YES;
  }

  UIImage *pinImage = [UIImage imageNamed:@"BluePin.png"];
  if (pinImage != nil){
    annotationView.image = pinImage;
  }

  result = annotationView;
```

```
  return result;
}
```

В данном коде отображаем картинку под названием `BluePin.png` (в пакете нашего приложения) для любого маркера, который ставится на карте. Определение реализации класса `MyAnnotation` приводится в разделе 9.5.

## Обсуждение

Объект делегата, относящийся к классу `MKMapView`, должен соответствовать протоколу `MKMapViewDelegate` и реализовывать метод `mapView:viewForAnnotation:`. Возвращаемое значение этого метода является экземпляром класса `MKAnnotationView`. Любой объект, являющийся подклассом вышеупомянутого класса, по умолчанию наследует свойство `image`. Если присвоить этому свойству такое значение, то мы заменим стандартное значение, предоставляемое во фреймворке Map Kit. Результат показан на рис. 9.4.

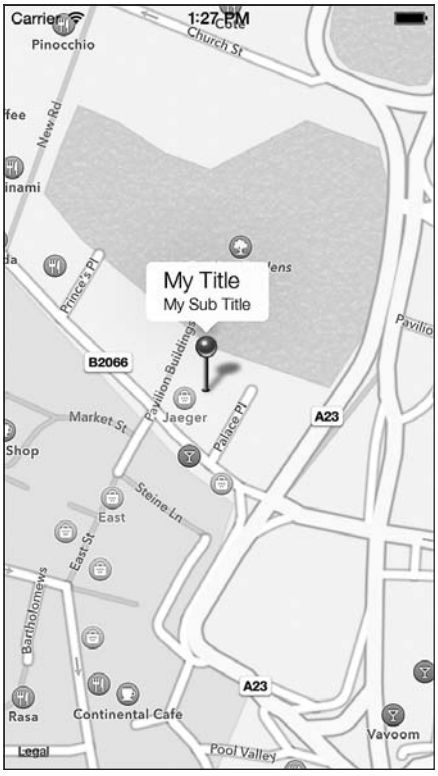


**Рис. 9.4.** Наше собственное изображение, показанное в картографическом виде

## См. также

Раздел 9.5.

# 9.7. Преобразование обычных адресов в данные широты и долготы

## Постановка задачи

Имеется адрес определенного места, необходимо найти его географические координаты (*широту* и *долготу*).

## Решение

Воспользуйтесь методом `geocodeAddressString:completionHandler:` из класса `CLGeocoder`.

## Обсуждение

*Обратное геокодирование* (Reverse Geocoding) — это процесс получения обычного адреса (то есть страны, города и т. д.) на базе известного пространственного расположения (координат широты и долготы). В свою очередь, *геокодирование* — это процесс нахождения пространственного расположения в сетке координат на основе известного адреса. Функции геокодирования и обратного геокодирования заключены в классе `CLGeocoder` фреймворка Core Location.

Пространственное местоположение геокодируется путем передачи адреса в формате `NSString` методу `geocodeAddressString:completionHandler:`, относящемуся к классу `CLGeocoder`. Параметр `completionHandler` этого метода принимает блоковый объект, не возвращающий никакого значения и имеющий два параметра:

- массив меток (типа `NSArray`). Метками будут обозначены те точки на карте, которые соответствуют критериям поискового запроса;
- ошибку (типа `NSError`), которая будет преобразована в код ошибки, если геокодирование не удастся.

Итак, сначала объявим свойство типа `CLGeocoder`:

```
#import "ViewController.h"
#import <CoreLocation/CoreLocation.h>

@interface ViewController ()
@property (nonatomic, strong) CLGeocoder *myGeocoder;
@end

@implementation ViewController
```

Идем дальше. Реализуем код для геокодирования адреса:

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

```
  /* У нас есть адрес. */
  NSString *oreillyAddress =
    @"1005 Gravenstein Highway North, Sebastopol, CA 95472, USA";

  self.myGeocoder = [[CLGeocoder alloc] init];

  [self.myGeocoder
   geocodeAddressString:oreillyAddress
   completionHandler:^(NSArray *placemarks, NSError *error) {

     if ([placemarks count] > 0 &&
         error == nil){
       NSLog(@"Found %lu placemark(s).", (unsigned long)[placemarks count]);
       CLPlacemark *firstPlacemark = [placemarks objectAtIndex:0];
       NSLog(@"Longitude = %f",
                    firstPlacemark.location.coordinate.longitude);
       NSLog(@"Latitude = %f", firstPlacemark.location.coordinate.latitude);
     }
     else if ([placemarks count] == 0 &&
              error == nil){
       NSLog(@"Found no placemarks.");
     }
     else if (error != nil){
       NSLog(@"An error occurred = %@", error);
     }

   }];

}
```

Как только программа будет запущена (даже в симуляторе), в окне консоли появятся следующие значения (при наличии активного сетевого соединения):

```
Found 1 placemark(s).
Longitude = -122.841135
Latitude = 38.410373
```

## См. также

Раздел 9.8.

# 9.8. Преобразование данных широты и долготы в обычные адреса

## Постановка задачи

Имеются значения широты и долготы определенной точки в пространстве. Необходимо получить ее адрес.

## Решение

Получение обычного адреса на основании известных пространственных координат (*x* и *y*) называется *обратным геокодированием*. Для выполнения такой операции нужно создать и использовать экземпляр класса `CLGeocoder`, а также предоставить блоковый объект завершения. При этом необходимо гарантировать, что блоковый объект не имеет возвращаемого значения и принимает два параметра:

- массив меток (типа `NSArray`). Метками будут обозначены те точки на карте, которые соответствуют критериям поискового запроса;
- ошибку (типа `NSError`), которая будет преобразована в код ошибки, если обратное геокодирование не удастся.

Инстанцировав объект типа `CLGeocoder`, мы используем его метод `reverseGeocodeLocation:completionHandler:` для выполнения обратного геокодирования.

Верхняя часть `.m`-файла простого контроллера вида, применяемого для этой цели, определяется следующим образом:

```
#import "ViewController.h"
#import <CoreLocation/CoreLocation.h>

@interface ViewController ()
@property (nonatomic, strong) CLGeocoder *myGeocoder;
@end

@implementation ViewController
```

В ходе загрузки вида можно выполнить обратное геокодирование:

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

  CLLocation *location = [[CLLocation alloc]
                          initWithLatitude:+38.4112810
                          longitude:-122.8409780f];

  self.myGeocoder = [[CLGeocoder alloc] init];

  [self.myGeocoder
   reverseGeocodeLocation:location
   completionHandler:^(NSArray *placemarks, NSError *error) {


if (error == nil && placemarks.count > 0){
CLPlacemark *placemark = placemarks[0];
       /* Результаты получены. */
       NSLog(@"Country = %@", placemark.country);
       NSLog(@"Postal Code = %@", placemark.postalCode);
       NSLog(@"Locality = %@", placemark.locality);
     }
     else if (error == nil &&
```

```
            [placemarks count] == 0){
      NSLog(@"No results were returned.");
    }
    else if (error != nil){
      NSLog(@"An error occurred = %@", error);
    }

  }];
}

- (void)viewDidUnload{
  [super viewDidUnload];
  self.myGeocoder = nil;
}
```

Если операция завершится успешно, то в массиве `placemarks` будут содержаться объекты типа `CLPlacemark`. Эти объекты будут отмечать адреса, удовлетворяющие значениям широты и долготы, которые мы сообщили методу `reverseGeocodeLocation:completionHandler:`. Итак, все, что от нас требуется, — убедиться в отсутствии ошибок и в том, что в массиве меток есть как минимум одна метка.

Методы `NSLog` из приведенного ранее кода выводят в окне консоли адрес, прошедший процедуру обратного геокодирования:

```
Country = United States
Postal Code = 95472
Locality = Sebastopol
```

## Обсуждение

В каждом приложении имеется лимит объема запросов на обратное геокодирование, которые могут быть выполнены в данном приложении за один день. Этот объем определяется провайдером серверного приложения, обеспечивающего поддержку геолокационных служб в iOS. Существуют различные платные онлайновые сервисы, которые предоставляют разработчикам сторонние API. Я не могу сейчас рекламировать какие-либо из подобных сервисов, но можете сами поискать их в Интернете, если захотите преодолеть ограничения, связанные с обратным геокодированием пространственных координат, существующие в настоящее время в iOS SDK. Чтобы выполнить запрос на обратное геокодирование, нужно создать экземпляр класса `CLGeocoder`. Этот класс требует активного сетевого соединения — оно необходимо для успешной обработки запросов. Значения, прошедшие обратное геокодирование, сообщаются блоку обработки завершения, который передается методу `reverseGeocodeLocation:completionHandler:`.

## См. также

Раздел 9.7.

# 9.9. Поиск в картографическом виде

## Постановка задачи

Требуется предоставить пользователям, просматривающим картографический вид, поисковую функцию. Например, можно помочь им найти все рестораны или тренажерные залы в конкретном регионе, отображенном на карте. Если пользователь находится в центре города и видит свое местоположение на карте, он может просто ввести в строку поиска слово «рестораны» — и приложение выполнит поиск по такому запросу.

## Решение

Картографические виды, без преувеличения, просто великолепны. Но иногда такой вид ничем не может помочь пользователю, который видит на экране просто одну большую карту. В таком случае вполне сгодится и обычная бумажная карта. Картографические возможности смартфонов интересны именно в контексте их интерактивности. Пользователь может находить на карте объекты, искать интересующие его места, получать информацию о том, как попасть в место, расположенное по тому или иному адресу. Apple включила в iOS SDK три очень удобных класса, позволяющих пользователю искать места на карте. Такой поиск совершенно прост. От вас требуется всего лишь ввести текстовый запрос о том, что вас интересует, например «рестораны» или «кафе, — и SDK выполнит за вас остальную работу. В этом разделе мы собираемся отобразить в контроллере вида картографический вид (с местоположением пользователя) и отслеживать местоположение пользователя. Таким образом, та точка, в которой он находится, всегда будет располагаться в центре карты.

Как только картографический вид поможет нам установить местоположение пользователя (предполагается, что пользователь разрешил нам это сделать), мы вызовем класс `MKLocalSearch` и выберем все рестораны, находящиеся поблизости от него. Первым делом определим картографический вид, вот так:

```
#import "ViewController.h"
#import <MapKit/MapKit.h>

@interface ViewController () <MKMapViewDelegate>
@property (nonatomic, strong) MKMapView *myMapView;
@end

@implementation ViewController
```

Далее необходимо создать картографический вид:

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

```
/* Создаем карту, совпадающую по размеру с нашим видом */
self.myMapView = [[MKMapView alloc]
initWithFrame:self.view.bounds];

self.myMapView.delegate = self;

/* Задаем для карты тип Standard */
self.myMapView.mapType = MKMapTypeStandard;

self.myMapView.autoresizingMask =
UIViewAutoresizingFlexibleWidth |
UIViewAutoresizingFlexibleHeight;

self.myMapView.showsUserLocation = YES;
self.myMapView.userTrackingMode = MKUserTrackingModeFollow;

/* Добавляем ее к нашему виду */
[self.view addSubview:self.myMapView];}
```

Мы используем свойство `showsUserLocation` картографического вида. Это логическое значение. Если оно равно `YES`, то картографический вид ищет местоположение пользователя (при наличии у нас разрешения на это). Все это, конечно, хорошо, но по умолчанию картографический вид действует так: он находит место на карте и отображает для него аннотацию, но не перемещает центральную точку карты и не увеличивает то место, где располагается пользователь. Иными словами, если в данный момент в картографическом виде отображается карта Великобритании, а пользователь находится где-то в Нью-Йорке, то он по-прежнему будет видеть на экране своего устройства карту Соединенного королевства. Чтобы исправить этот недостаток, нужно задать для свойства `userTrackingMode` картографического вида значение `MKUserTrackingModeFollow`, при котором центр картографического вида всегда соответствует местоположению пользователя. Отображаемая часть карты корректируется в соответствии с перемещением пользователя.

Теперь, когда мы приказали картографическому виду отслеживать местоположение пользователя, необходимо реализовать следующие методы делегатов картографического вида:

- `mapView:didFailToLocateUserWithError:` — вызывается в делегате, когда картографическому виду не удается определить местоположение пользователя. В этом методе мы выводим для пользователя предупреждение о том, что определить его местоположение не получается;
- `mapView:didUpdateUserLocation:` — вызывается в делегате картографического вида всякий раз, когда информация о местоположении пользователя обновляется. Таким образом, он всегда соответствует успешному варианту развития бизнес-логики. В этом методе можем реализовать локальную функцию поиска.

Сначала давайте реализуем метод `mapView:didFailToLocateUserWithError:`:

```
- (void) mapView:(MKMapView *)mapView
  didFailToLocateUserWithError:(NSError *)error{
    UIAlertView *alertView = [[UIAlertView alloc]
```

```
                                initWithTitle:@"Failed"
                                message:@"Could not get the user's location"
                                delegate:nil cancelButtonTitle:@"OK"
                                otherButtonTitles:nil];
    [alertView show];
}
```

Элементарно. Переходим к методу `mapView:didUpdateUserLocation:`:

```
- (void) mapView:(MKMapView *)mapView
  didUpdateUserLocation:(MKUserLocation *)userLocation{

    MKLocalSearchRequest *request = [[MKLocalSearchRequest alloc] init];
    request.naturalLanguageQuery = @"restaurants";

    MKCoordinateSpan span = MKCoordinateSpanMake(0.01, 0.01);

    request.region =
    MKCoordinateRegionMake(userLocation.location.coordinate, span);

    MKLocalSearch *search = [[MKLocalSearch alloc] initWithRequest:request];

    [search startWithCompletionHandler:
    ^(MKLocalSearchResponse *response, NSError *error) {

    for (MKMapItem *item in response.mapItems){
      NSLog(@"Item name = %@", item.name);
      NSLog(@"Item phone number = %@", item.phoneNumber);
      NSLog(@"Item url = %@", item.url);
      NSLog(@"Item location = %@", item.placemark.location);

    }

  }];

}
```

В этом методе все просто. Мы создаем локальный поисковый запрос и устанавливаем в качестве значения его свойства `naturalLanguageQuery` те элементы, которые мы хотим найти на карте, — в данном случае рестораны. Затем получаем местоположение пользователя и создаем на его основе регион типа `MKCoordinateRegion`. Мы делаем это потому, что хотим определить область, окружающую пользователя, и выполнить поиск в этой области. Область сообщает движку поиска местоположения о том, что мы хотим ограничить круг поиска заданным регионом. Как только регион создан, задаем его в качестве значения свойства `region` для локального поиска. Сделав это, можно приступать к поиску. Для этого мы отправляем локальный поисковый запрос методу экземпляра `startWithCompletionHandler:`, относящемуся к классу `MKLocalSearch`. Этот метод принимает блок в качестве параметра. Данный блок кода вызывается при поступлении результатов поиска или возникновении ошибки.

Найденные элементы будут записаны в свойстве `mapItems` параметра отклика нашего блокового объекта, эти картографические элементы будут относиться к типу `MKMapItem`. У каждого элемента будут свойства — в частности, `name`, `phoneNumber` и `url` — которые помогут нанести на карту интересующие нас точки. При этом мы воспользуемся приемами, изученными ранее в этой главе, — например, отобразим на карте маркеры, о которых говорили в разделе 9.4.

## См. также

Разделы 9.4–9.6.

# 9.10. Отображение направлений на карте

## Постановка задачи

Необходимо отображать на карте направления, подсказывая таким образом пользователю, как попасть из точки A в точку B.

## Решение

Инстанцируйте объект типа `MKDirections` и вызовите метод экземпляра `calculateDirectionsWithCompletionHandler:`, относящийся к этому объекту. Так будет вызван обработчик завершения, а вам будет передан объект типа `MKDirectionsResponse`. Воспользуйтесь таким откликом с информацией о направлениях, чтобы открыть на устройстве приложение Maps (Карты). Этому мы также вскоре научимся.

## Обсуждение

Вы можете отображать на экране направления, подсказывающие пользователю, как пройти или проехать куда-либо. Но такая возможность доступна только в приложении Maps (Карты). Соответственно, вы не сможете наносить такие линии на карту прямо в картографическом виде внутри приложения. Способ указания направлений на карте в приложении Maps очень прост. Чтобы создать на экране такие линии, потребуется инстанцировать экземпляр класса `MKDirections`. Для работы с этим классом нужен уже готовый экземпляр `MKDirectionsRequest`.

Кроме того, для создания запроса на отображение направлений потребуется создать экземпляры `MKMapItem`. Каждый из таких элементов будет соответствовать точке на карте. Суть такова: если вы хотите отобразить на карте направления, помогающие пользователю найти путь из точки A в точку B, то эти точки потребуется представить в виде элементов карты. На базе информации об этих элементах создается запрос, а затем для получения направлений используется класс `MKDirections`. После получения направлений можно поступить двумя способами.

- Обработать направления самостоятельно. Например, с помощью одной из техник, изученных ранее в этой главе (см. раздел 9.4), вы можете получить все автозаправки (их метки), расположенные по пути из точки A в точку B, а затем снабдить эти точки на карте маркерами.
- Отправить информацию о направлениях в приложение Maps (Карты) для отображения.

В данном разделе мы исследуем второй вариант. Итак, предположим, что мы хотим показать *направления проезда* от той точки, в которой сейчас находимся, в другую произвольную точку на карте. В этом разделе мы задаем следующий адрес назначения: Churchill Square Shopping Center, Brighton, United Kingdom (Торговый центр «Черчилль», Брайтон, Соединенное королевство). С помощью технологии, изученной в разделе 9.7, мы сможем преобразовать обычный адрес, выразив его в координатах широты и долготы. Затем воспользуемся этой информацией для создания экземпляра класса `MKPlacemark` — подробнее об этом в дальнейшем.

Итак, начнем. Первым делом потребуется импортировать фреймворк Core Location, с помощью которого мы сможем преобразовать вышеупомянутый адрес в географические координаты (широту и долготу). Кроме того, импортируем фреймворк MapKit, с помощью которого сможем создать запрос направления. При помощи модулей, работу с которыми обеспечивает LLVM, мы без труда импортируем эти фреймворки в приложение:

```
#import "AppDelegate.h"
#import <CoreLocation/CoreLocation.h>
#import <MapKit/MapKit.h>

@implementation AppDelegate

<# Оставшаяся часть вашего кода находится здесь #>
```

Далее воспользуемся информацией, изученной в разделе 9.7, и преобразуем адрес в данные широты и долготы:

```
- (BOOL)            application:(UIApplication *)application
  didFinishLaunchingWithOptions:(NSDictionary *)launchOptions{

    NSString *destination = @"Churchill Square Shopping Center, \
    Brighton, United Kingdom";

    [[CLGeocoder new]
     geocodeAddressString:destination
     completionHandler:^(NSArray *placemarks, NSError *error) {

       <# Теперь у нас есть координаты адреса #>

    }];

    self.window = [[UIWindow alloc] initWithFrame:[[UIScreen mainScreen] bounds]];
    // Точка переопределения для дополнительной настройки после запуска приложения.
    self.window.backgroundColor = [UIColor whiteColor];
```

```
    [self.window makeKeyAndVisible];
    return YES;
}
```

Весь код, приведенный далее в этом разделе, будет находиться в объекте блока завершения, относящемся к методу geocodeAddressString:completionHandler: только что написанного нами класса CLGeocoder.

Блок завершения будет давать ссылку на объект ошибки. Вам потребуется считать этот объект ошибки и, если ошибка вернется, обработать ее соответствующим образом. Итак, давайте сообщим MapKit, что в качестве точки отсчета всех направлений должен использоваться тот пункт, в котором мы сейчас находимся. Для создания запроса направлений мы воспользуемся классом `MKDirectionsRequest`, а в качестве значения свойства `source` этого запроса зададим значение метода класса `mapItemForCurrentLocation` (этот метод относится к классу `MKMapItem`):

```
if (error != nil){
  /* Здесь обрабатываем ошибку, например отобразив окно с предупреждением */
  return;
}

MKDirectionsRequest *request = [[MKDirectionsRequest alloc] init];
request.source = [MKMapItem mapItemForCurrentLocation];
```

Ранее мы создали строковый объект, в котором содержался наш адрес назначения. Теперь у нас есть экземпляр `CLPlacemark` и нужно преобразовать его в экземпляр `MKPlacemark`, который можно будет задать в запросе направления как значение свойства `Destination`:

```
/* Преобразуем метку назначения CoreLocation в метку MapKit */

/* Получаем метку адреса назначения */
CLPlacemark *placemark = placemarks[0];
CLLocationCoordinate2D destinationCoordinates =
placemark.location.coordinate;
MKPlacemark *destination = [[MKPlacemark alloc]
                            initWithCoordinate:destinationCoordinates
                            addressDictionary:nil];

request.destination = [[MKMapItem alloc]
                       initWithPlacemark:destination];
```

В классе `MKDirectionsRequest` есть свойство `transportType`, относящееся к типу `MKDirectionsTransportType`:

```
typedef NS_OPTIONS(NSUInteger, MKDirectionsTransportType) {
  MKDirectionsTransportTypeAutomobile = 1 << 0,
  MKDirectionsTransportTypeWalking = 1 << 1,
  MKDirectionsTransportTypeAny = 0x0FFFFFFF
} NS_ENUM_AVAILABLE(10_9, 7_0);
```

Поскольку мы хотим отобразить направления проезда из исходной точки в точку назначения, в этом разделе воспользуемся значением MKDirectionsTransportTypeAutomobile:

```
/* Мы собираемся попасть в точку назначения на автомобиле */
request.transportType = MKDirectionsTransportTypeAutomobile;
```

Наконец, создаем экземпляр класса MKDirections с помощью метода-инициализатора initWithRequest:. В качестве параметра инициализатор принимает экземпляр класса MKDirectionsRequest. Мы уже создали и подготовили этот объект с элементом карты, указывающим точку отправления и точку назначения.

Затем применим в нашем классе, описывающем направления, метод экземпляра calculateDirectionsWithCompletionHandler:. Этот метод позволяет получить направления от исходной точки к точке назначения. В качестве параметра этот метод принимает блоковый объект, предоставляющий нам объект типа MKDirectionsResponse и ошибку типа NSError (эта сущность позволяет определить, не произошла ли ошибка). У объекта отклика, который будет нам передан, есть два очень важных свойства: source и destination. Они будут соответствовать тем элементам карты (начальной и конечной точкам), которые мы задали ранее. Будучи в этом блоке, можно либо просто взять отклик с точкой назначения и обработать его вручную (как уже объяснялось), либо передать информацию о начальной и конечной точках в приложение Maps (Карты) для отображения, вот так:

```
/* Получаем направления */
MKDirections *directions = [[MKDirections alloc]
                            initWithRequest:request];

[directions calculateDirectionsWithCompletionHandler:
 ^(MKDirectionsResponse *response, NSError *error) {

  /* Можно вручную выполнить синтаксический разбор отклика, но здесь мы
   поступим иначе и воспользуемся приложением Maps (Карты) для отображения
   начальной и конечной точек. Делать такой вызов API не обязательно, так как
   ранее мы уже подготовили элементы карты. Но здесь вызов делается
   в демонстрационных целях. Мы показываем, что в отклике с направлениями
   содержится не только информация о начальной и конечной точках */

  /* Отображаем направления в приложении Maps */
  [MKMapItem
   openMapsWithItems:@[response.source, response.destination]
   launchOptions:@{
                   MKLaunchOptionsDirectionsModeKey :
                     MKLaunchOptionsDirectionsModeDriving}];
  }];
```

Теперь, если объединить весь написанный код, он получится довольно компактным:

```
#import "AppDelegate.h"
#import <CoreLocation/CoreLocation.h>
#import <MapKit/MapKit.h>
```

```
@implementation AppDelegate

- (BOOL)            application:(UIApplication *)application
  didFinishLaunchingWithOptions:(NSDictionary *)launchOptions{

    NSString *destination = <# Place your destination address here #>;

    [[CLGeocoder new]
     geocodeAddressString:destination
     completionHandler:^(NSArray *placemarks, NSError *error) {
    if (error != nil){
      /* Здесь обрабатываем ошибку, например отобразив окно
       с предупреждением */
      return;
    }

    MKDirectionsRequest *request = [[MKDirectionsRequest alloc] init];
    request.source = [MKMapItem mapItemForCurrentLocation];

    /* Преобразуем метку назначения CoreLocation в метку MapKit */
    /* Получаем метку адреса назначения*/
    CLPlacemark *placemark = placemarks[0];
    CLLocationCoordinate2D destinationCoordinates =
    placemark.location.coordinate;
    MKPlacemark *destination = [[MKPlacemark alloc]
                                initWithCoordinate:destinationCoordinates
    addressDictionary:nil];

    request.destination = [[MKMapItem alloc]
                           initWithPlacemark:destination];

    /* Мы собираемся попасть в точку назначения на автомобиле */
    request.transportType = MKDirectionsTransportTypeAutomobile;

    /* Получаем направления */
    MKDirections *directions = [[MKDirections alloc]
                                initWithRequest:request];
    [directions calculateDirectionsWithCompletionHandler:
     ^(MKDirectionsResponse *response, NSError *error) {

    /* Можно вручную выполнить синтаксический разбор отклика, но здесь мы
     поступим иначе и воспользуемся приложением Maps (Карты) для отображения
     начальной и конечной точек. Делать такой вызов API необязательно,
     так как ранее мы уже подготовили элементы карты. Но здесь вызов
     делается в демонстрационных целях. Мы показываем, что в отклике
     с направлениями содержится не только информация о начальной и конечной
     точках */
    /* Отображаем направления в приложении Maps */
    [MKMapItem
     openMapsWithItems:@[response.source, response.destination]
         launchOptions:@{
```

```
                                MKLaunchOptionsDirectionsModeKey :
                                  MKLaunchOptionsDirectionsModeDriving}];
    }];
  }];

  self.window = [[UIWindow alloc]
                 initWithFrame:[[UIScreen mainScreen] bounds]];
  // Точка переопределения для дополнительной настройки после запуска приложения
  self.window.backgroundColor = [UIColor whiteColor];
  [self.window makeKeyAndVisible];
  return YES;
}
```

Я запущу это приложение в симуляторе iOS, так как выбранная мной конечная точка находится слишком близко от того места, где я нахожусь (начальной точки). Результат получится примерно таким, как на рис. 9.5.

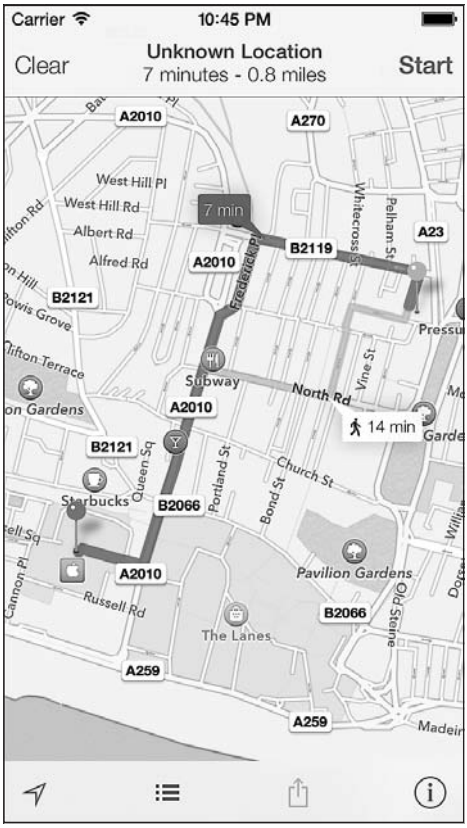


**Рис. 9.5**. Отображение направлений на карте

## См. также

Раздел 9.3.

# 10 Реализация распознавания жестов

## 10.0. Введение

Жест (Gesture) — это комбинация событий касания. Жесты применяются, например, в стандартном приложении Photo (Фото) для iOS. В этой программе пользователь может увеличивать или уменьшать фотографию, двигая двумя пальцами в разные стороны или навстречу друг другу. Некоторые образцы кода, чаще всего применяемого для обнаружения событий, связанных с жестикуляцией, инкапсулированы во встроенные классы iOS SDK, которые пригодны для многократного использования. Эти классы можно применять для обнаружения смахивания (Swipe), щипка (Pinch), панорамирования (Pan), нажатия (Tap), перетаскивания (Drag), долгого нажатия (Long Press) и вращения (Rotation).

Распознаватели жестов необходимо добавлять к экземплярам класса `UIView`. Один вид может быть связан с несколькими распознавателями жестов. Как только вид регистрирует жест, при необходимости он должен будет передать данный жест другим видам, расположенным ниже в иерархической цепочке.

Некоторые события, возникающие при работе приложения, могут быть сложны для обработки и требовать, чтобы одно и то же событие обнаруживалось в разных видах отдельно взятого приложения. Таким образом, возникает необходимость в распознавателях жестов, пригодных для многократного использования. В iOS SDK 5 интегрированы распознаватели шести жестов, таких как:

- смахивание;
- вращение;
- щипок;
- панорамирование;
- длинное нажатие;
- нажатие.

Общий принцип обработки жестов с помощью этих встроенных распознавателей таков.

1. Для требуемого распознавателя жестов создается объект данных нужного типа.
2. Этот объект добавляется в качестве распознавателя жестов к тому виду, который будет принимать жесты.
3. Пишется метод, вызываемый при возникновении жеста и осуществляющий указанное вами действие.

Метод, который ассоциируется в качестве целевого метода с любым распознавателем жестов, должен следовать перечисленным далее правилам:

- возвращать `void`;
- либо не принимать параметров, либо принимать единственный параметр типа `UIGestureRecognizer`, в котором система будет передавать распознаватель жестов, вызывающий данный метод.

Рассмотрим два примера:

```
- (void) tapRecognizer:(UITapGestureRecognizer *)paramSender{
  /* */
}

- (void) tapRecognizer{
  /* */
}
```

Распознаватели жестов делятся на две категории: *дискретные* (Discrete) и *непрерывные* (Continuous). Дискретные распознаватели жестов регистрируют связанные с ними события жестов, а после этого вызывают метод в своем обладателе. Непрерывные распознаватели жестов сообщают своему объекту-обладателю о жесте на протяжении всего того времени, пока этот жест осуществляется, и многократно вызывают метод в своем целевом объекте, пока это событие не закончится.

Например, событие двойного нажатия является дискретным. Хотя оно и состоит из двух нажатий, система улавливает, что промежуток между ними был очень кратким и оба нажатия можно воспринимать как единое событие. Распознаватель двойного нажатия вызывает в своем целевом объекте соответствующий метод, как только будет зарегистрировано двойное нажатие.

Вращение, напротив, обрабатывается непрерывным распознавателем жестов. Как только пользователь начинает вращательный жест, начинается и работа распознавателя, а оканчивается этот жест, только когда пользователь отрывает пальцы от экрана. Метод, предоставляемый классу распознавателя вращательных жестов, вызывается с краткими интервалами до тех пор, пока событие не завершится.

Распознаватели жестов можно добавлять к любому экземпляру класса `UIView` с помощью метода `addGestureRecognizer:`, относящегося к виду. При необходимости распознаватели можно удалять, пользуясь методом `removeGestureRecognizer:`.

У класса `UIGestureRecognizer` есть свойство под названием `state`. Свойство `state` представляет различные состояния распознавателя жестов, которые он принимает в ходе распознавания. Последовательности претерпеваемых состояний различаются у дискретных и непрерывных распознавателей жестов.

Дискретный распознаватель жестов может проходить через три следующих состояния:

- UIGestureRecognizerStatePossible;
- UIGestureRecognizerStateRecognized;
- UIGestureRecognizerStateFailed.

В зависимости от ситуации дискретный распознаватель жестов может сообщать своей цели о состоянии UIGestureRecognizerStateRecognized либо о состоянии UIGestureRecognizerStateFailed, если в процессе распознавания возникнет ошибка.

Непрерывные распознаватели жестов претерпевают иную серию состояний, которые посылают своим целям:

- UIGestureRecognizerStatePossible;
- UIGestureRecognizerStateBegan;
- UIGestureRecognizerStateChanged;
- UIGestureRecognizerStateEnded;
- UIGestureRecognizerStateFailed.

Состояние распознавателя жестов меняется на UIGestureRecognizerStatePossible в том случае, когда распознаватель собирает в виде информацию о событиях касаний и в любой момент может обнаружить интересующий его жест. Кроме вышеупомянутых состояний непрерывного распознавателя жестов может возникать и состояние UIGestureRecognizerStateCancelled, если жест по какой-то причине прерывается. Например, жест панорамирования может быть прерван входящим телефонным вызовом. В данном случае распознаватель жестов перейдет в состояние UIGestureRecognizerStateCancelled и перестанет отправлять объекту-получателю какие-либо сообщения, если пользователь не повторит всю жестовую последовательность.

Опять же, если непрерывный распознаватель жестов столкнется с ситуацией, которую не удается разрешить с помощью имеющихся у системы возможностей, возникнет состояние UIGestureRecognizerStateFailed, а не UIGestureRecognizer-StateEnded.

# 10.1. Обнаружение жестов смахивания

## Постановка задачи

Необходимо идентифицировать скользящие жесты смахивания, которые пользователь осуществляет на виде, например, когда убирает картинку с окна.

## Решение

Инстанцируйте объект типа UISwipeGestureRecognizer и добавьте его к экземпляру UIView:

```
#import "ViewController.h"

@interface ViewController ()
@property (nonatomic, strong)
```

```
UISwipeGestureRecognizer *swipeGestureRecognizer;
@end

@implementation ViewController

- (void)viewDidLoad {
  [super viewDidLoad];

  /* Инстанцируем объект. */
  self.swipeGestureRecognizer = [[UISwipeGestureRecognizer alloc]
                                 initWithTarget:self
                                 action:@selector(handleSwipes:)];

  /* Необходимо обнаруживать жесты смахивания,
     направленные справа налево. */
  self.swipeGestureRecognizer.direction =
    UISwipeGestureRecognizerDirectionLeft;

  /* Нужен только один палец. */
  self.swipeGestureRecognizer.numberOfTouchesRequired = 1;

  /* Добавляем к виду. */
  [self.view addGestureRecognizer:self.swipeGestureRecognizer];

}
```

Распознаватель жестов может быть создан как автономный объект, но в данном случае, поскольку мы используем распознаватель только с одним видом, мы запрограммировали его как свойство контроллера вида, который будет принимать жест (`self.swipeGestureRecognizer`). В подразделе «Обсуждение» данного раздела показано применение в этом коде метода `handleSwipes:`, выступающего в качестве цели для распознавателя жестов смахивания.

## Обсуждение

Жест смахивания (скольжения) — одно из наиболее простых движений, регистрируемых встроенными распознавателями жестов, входящими в состав iOS SDK. Это простое движение одного или нескольких пальцев по экрану в том или ином направлении. Класс `UISwipeGestureRecognizer`, как и любые другие распознаватели жестов, наследует от класса `UIGestureRecognizer` и добавляет к нему различные функции. В частности, это свойства, позволяющие указывать направление, в котором должны выполняться жесты смахивания, чтобы система их обнаружила, а также определять, сколько пальцев пользователь должен держать на экране, чтобы можно было совершить жест смахивания. Не забывайте, что жесты смахивания являются дискретными.

Метод `handleSwipes:`, которым мы воспользовались при написании распознавателя жестов, можно реализовать следующим образом:

```
- (void) handleSwipes:(UISwipeGestureRecognizer *)paramSender{

  if (paramSender.direction & UISwipeGestureRecognizerDirectionDown){
```

```
    NSLog(@"Swiped Down.");
  }
  if (paramSender.direction & UISwipeGestureRecognizerDirectionLeft){
    NSLog(@"Swiped Left.");
  }
  if (paramSender.direction & UISwipeGestureRecognizerDirectionRight){
    NSLog(@"Swiped Right.");
  }
  if (paramSender.direction & UISwipeGestureRecognizerDirectionUp){
    NSLog(@"Swiped Up.");
  }

}
```

В свойстве direction экземпляра класса UISwipeGestureRecognizer можно скомбинировать несколько направлений, пользуясь поразрядным операндом OR (ИЛИ). В языке Objective-C он обозначается вертикальной чертой (|). Например, чтобы получить прямое диагональное смахивание по направлению к нижнему левому углу экрана, можно скомбинировать значения UISwipeGestureRecognizerDirectionLeft и UISwipeGestureRecognizerDirectionDown, применяя при создании распознавателя жестов знаки вертикальной черты. В данном примере мы пытаемся обнаружить только жесты смахивания, идущие справа налево.

Обычно смахивание выполняется только одним пальцем, но существует свойство `numberOfTouchesRequired` класса `UISwipeGestureRecognizer`, в котором можно указать количество пальцев, необходимое для того, чтобы жест был распознан.

# 10.2. Обнаружение жестов вращения

## Постановка задачи

Необходимо обнаруживать, когда пользователь пытается повернуть пальцами элемент, изображенный на экране.

## Решение

Создайте экземпляр класса `UIRotationGestureRecognizer` и присоедините его к целевому виду:

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

  self.helloWorldLabel = [[UILabel alloc] initWithFrame:CGRectZero];
  self.helloWorldLabel.text = @"Hello, World!";
  self.helloWorldLabel.font = [UIFont systemFontOfSize:16.0f];
  [self.helloWorldLabel sizeToFit];
  self.helloWorldLabel.center = self.view.center;
  [self.view addSubview:self.helloWorldLabel];
```

```
  self.rotationGestureRecognizer = [[UIRotationGestureRecognizer alloc]
                                    initWithTarget:self
                                    action:@selector(handleRotations:)];

  [self.view addGestureRecognizer:self.rotationGestureRecognizer];

}
```

## Обсуждение

Распознаватель жестов `UIRotationGestureRecognizer`, как понятно из его названия, отлично подходит для распознавания жестов вращения и помогает делать пользовательские интерфейсы значительно более интуитивно понятными. Например, если пользователь работает с устройством в полноэкранном режиме и встречает на экране какое-то изображение, ориентация которого не соответствует ориентации экрана, то вполне логично, что он попытается подправить картинку, повернув ее на дисплее.

Класс `UIRotationGestureRecognizer` реализует свойство `rotation`, указывающее степень и направление вращения, заданного жестом пользователя. Эти показатели выражаются в радианах. Вращение определяется в зависимости от исходного положения пальцев (`UIGestureRecognizerStateBegan`) и их конечного положения (`UIGestureRecognizerStateEnded`).

Чтобы вращать элементы пользовательского интерфейса, наследующие от класса `UIView`, можно передать свойство `rotation` распознавателя жестов вращения функции `CGAffineTransformMakeRotation`, чтобы она выполнила аффинное преобразование, как показано в следующем примере.

Код, приведенный в подразделе «Решение» данного раздела, передает актуальный объект (в данном случае контроллер вида) к цели распознавателя жестов вращения. Целевой селектор задается как `handleRotations:` — метод, который мы хотим реализовать. Но прежде, чем приступить к этому, изучим заголовочный файл контроллера вида:

```
#import "ViewController.h"
@interface ViewController ()

@property (nonatomic, strong)
  UIRotationGestureRecognizer *rotationGestureRecognizer;

@property (nonatomic, strong)
  UILabel *helloWorldLabel;
/* Из этого объявления свойства можно удалить метки nonatomic
   и unsafe_unretained. Имея значение с плавающей точкой, компилятор
   автоматически сгенерирует для нас обе эти метки. */
@property (nonatomic, unsafe_unretained)
  CGFloat rotationAngleInRadians;

@end
@implementation ViewController
```

Прежде чем продолжать, рассмотрим, за что отвечает каждое из этих свойств и почему они объявляются:

- `helloWorldLabel` — это метка, которую мы должны поставить на виде в контроллере вида. Потом напишем код для вращения этой метки. Вращение будет начинаться всякий раз, когда пользователь станет совершать вращательные жесты на виде, обладающем этой меткой (в данном случае речь идет о виде нашего контроллера вида);
- `rotationGestureRecognizer` — это экземпляр распознавателя жестов вращения, который мы позже выделим и инициализируем;
- `rotationAngleInRadians` — это значение, которое будет запрашиваться как точный угол поворота метки. Изначально это свойство устанавливается в нуль. Поскольку углы вращения, сообщаемые распознавателем, сбрасываются перед каждым новым пуском распознавателя, можно всякий раз сохранять значение распознавателя жестов вращения, когда он переходит в состояние `UIGestureRecognizerStateEnded`. В следующий раз при запуске жеста мы суммируем предыдущее значение вращения и новое значение вращения, получив в результате общий угол вращения.

Размер метки и ее центральная точка могут выбираться произвольно. Аналогично непринципиально и положение самой метки, так как мы просто пытаемся вращать метку вокруг ее центра независимо от того, в какой части вида она расположена. Единственный важный аспект здесь заключается в том, что в универсальных приложениях положение метки в контроллере вида следует рассчитывать динамически при работе с разными целями (то есть устройствами), основываясь на размерах ее родительского вида. Если приложение запускается на иных устройствах, кроме iPhone и iPad, метка может оказываться в различных точках экрана.

Применяя свойство метки `center` и устанавливая эту точку в центре объемлющего вида, мы выравниваем по центру и содержимое самой метки. Преобразование вращения, которое мы применим к данной метке, станет вращать метку вокруг ее центральной точки. А если содержимое метки выровнено по левому или по правому краю и ее истинный контур шире, чем пространство, необходимое для полного отображения содержимого (без отсечения), то вращаться такая метка будет довольно неестественно и не вокруг центра. Если вам любопытно, как это выглядит на практике, попробуйте выровнять содержимое метки по левому или правому краю и посмотрите, что получится.

Как показано в подразделе «Решение» данного раздела, созданный нами распознаватель жестов вращения будет отправлять свои события методу `handleRotations:`. Вот реализация этого метода:

```
- (void) handleRotations:(UIRotationGestureRecognizer *)paramSender{

  if (self.helloWorldLabel == nil){
    return;
  }

  /* Берем предыдущее значение вращения и суммируем его с актуальным
    значением вращения. */
```

```
  self.helloWorldLabel.transform =
  CGAffineTransformMakeRotation(self.rotationAngleInRadians +
                                paramSender.rotation);

  /* Когда значение завершится, сохраняем полученный угол для
     дальнейшего использования. */
  if (paramSender.state == UIGestureRecognizerStateEnded){
    self.rotationAngleInRadians += paramSender.rotation;
  }

}
```

Распознаватель жестов вращения посылает нам информацию об углах вращения очень интересным способом. Этот распознаватель является непрерывным. Это означает, что распознаватель приступает к вычислению углов, как только пользователь начинает жест вращения, и отправляет обновления методу-обработчику с краткими интервалами до тех пор, пока пользователь не закончит жест. В каждом сообщении начальный угол воспринимается как нулевой, и это сообщение содержит информацию о начальной точке вращения (достигнутой после окончания предыдущего акта вращения) и конечной точке. Таким образом, полный эффект от данного жеста можно узнать, только суммировав значения углов, полученные в результате различных событий. Движение по часовой стрелке дает положительное угловое значение, а против часовой стрелки — отрицательное.

Если вы работаете с симулятором iPhone, а не с реальным устройством, то можете имитировать и вращательное движение. Для этого нужно просто удерживать клавишу Option. В симуляторе вы увидите два кружка, которые появятся на одинаковом расстоянии от центра экрана. Они будут соответствовать подушечкам двух пальцев. Если вы захотите переместить «пальцы» из центра в другую точку экрана, то нужно нажать клавишу Shift, удерживая Alt, и перейти в желаемую точку. Когда вы отпустите указатель, новая точка станет центром для двух подушечек пальцев.

Теперь мы просто присвоим этот угол углу вращения метки. Но вы можете представить, что произойдет, когда одно вращение закончится, а другое начнется? Угол второго вращательного жеста заменит первое вращение в значении `rotation` и будет сообщен обработчику. Поэтому, как только вращательный жест завершится, необходимо сохранить текущее вращательное значение метки. Угловое значение, получаемое в результате каждого вращательного движения, должно суммироваться с предыдущими по очереди. Ранее было показано, как этот результат присваивается общему вращательному преобразованию метки.

Кроме того, прежде мы говорили о применении функции `CGAffineTransformMakeRotation` для создания аффинного преобразования. Функции iOS SDK, названия которых начинаются на CG, относятся к фреймворку Core Graphics. Чтобы в программах, использующих Core Graphics, успешно протекали процессы компиляции и связывания, необходимо убедиться, что Core Graphics добавлен в список фреймворков. В новых версиях Xcode стандартный проект автоматически связывается с фреймворком Core Graphics, так что об этом можно не беспокоиться.

Теперь, когда вы уверены, что фреймворк Core Graphics добавлен к целевой сборке, можно скомпилировать и запустить приложение.

## См. также

Раздел 10.6.

# 10.3. Обнаружение жестов панорамирования и перетаскивания

## Постановка задачи

Требуется предоставить пользователю возможность перемещать элементы в пользовательском интерфейсе, касаясь сенсорного экрана пальцами.

Жесты панорамирования — это непрерывные движения пальцев по экрану. Напоминаю, что жесты смахивания являются дискретными. Это означает, что метод, задаваемый для распознавателя жестов панорамирования в качестве целевого, вызывается многократно от начала и до конца процесса распознавания.

## Решение

Воспользуйтесь классом `UIPanGestureRecognizer`:

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

  /* Сначала создаем метку. */
  CGRect labelFrame = CGRectMake(0.0f,    /* X */
                                 0.0f,    /* Y */
                                 150.0f,  /* Ширина */
                                 100.0f); /* Высота */

  self.helloWorldLabel = [[UILabel alloc] initWithFrame:labelFrame];
  self.helloWorldLabel.text = @"Hello World";
  self.helloWorldLabel.backgroundColor = [UIColor blackColor];
  self.helloWorldLabel.textColor = [UIColor whiteColor];
  self.helloWorldLabel.textAlignment = UITextAlignmentCenter;

  /* Убеждаемся, что мы активизировали пользовательские взаимодействия;
     в противном случае эта метка не будет фиксировать события нажатия. */
  self.helloWorldLabel.userInteractionEnabled = YES;

  /* А теперь убеждаемся, что метка отображается в виде. */
  [self.view addSubview:self.helloWorldLabel];

  /* Создаем распознаватель жестов панорамирования. */
```

```
  self.panGestureRecognizer = [[UIPanGestureRecognizer alloc]
                               initWithTarget:self
                               action:@selector(handlePanGestures:)];

  /* Для активизации распознавателя жестов панорамирования требуется
     один палец. */
  self.panGestureRecognizer.minimumNumberOfTouches = 1;
  self.panGestureRecognizer.maximumNumberOfTouches = 1;

  /* Добавляем распознаватель к виду. */
  [self.helloWorldLabel addGestureRecognizer:self.panGestureRecognizer];

}
```

Распознаватель жестов панорамирования будет вызывать метод `handlePanGestures:` в качестве целевого. Этот метод описан в подразделе «Решение» данного раздела.

## Обсуждение

Распознаватель `UIPanGestureRecognizer`, как понятно из его названия[1], способен обнаруживать жесты *панорамирования*. В ходе работы этот распознаватель проходит через следующие состояния:

- `UIGestureRecognizerStateBegan`;
- `UIGestureRecognizerStateChanged`;
- `UIGestureRecognizerStateEnded`.

Целевой метод этого распознавателя жестов можно реализовать следующим образом. Приведенный код будет непрерывно перемещать центр метки вслед за пальцем пользователя, и на протяжении этого процесса будет сообщаться о событиях `GestureRecognizerStateChanged`:

```
- (void) handlePanGestures:(UIPanGestureRecognizer*)paramSender{

  if (paramSender.state != UIGestureRecognizerStateEnded &&
      paramSender.state != UIGestureRecognizerStateFailed){
      CGPoint location = [paramSender
                          locationInView:paramSender.view.superview];
      paramSender.view.center = location;
  }

}
```

Чтобы можно было перемещать метку вида, относящегося к контроллеру вида, нам нужно знать не положение метки, а положение пальца на виде. Поэтому мы вызываем метод locationInView: распознавателя жестов панорамирования и передаем родительский вид метки в качестве целевого.

[1] Pan (англ.) — «панорамирование», recognizer (англ.) — «распознаватель». — *Примеч. пер.*

Воспользуйтесь методом `locationInView:` распознавателя жестов панорамирования, чтобы найти позиции пальцев (или пальца), которые в настоящее время совершают этот жест. Чтобы найти положение нескольких пальцев, пользуйтесь методом `locationOfTouch:inView:`. С помощью свойств `minimumNumberOfTouches` и `maximumNumberOfTouches` класса `UIPanGestureRecognizer` можно одновременно регистрировать более одного панорамирующего касания. Но в примере ради простоты мы пытаемся найти положение всего одного пальца.

В состоянии UIGestureRecognizerStateEnded сообщаемые значения *x* и *y* могут быть и нечисловыми — они могут равняться NAN. Вот почему необходимо избегать использования значений, сообщаемых именно в этом конкретном состоянии.

# 10.4. Обнаружение жестов долгого нажатия

## Постановка задачи

Необходимо обнаруживать ситуации, в которых пользователь нажимает определенный экранный элемент и удерживает палец на экране в течение некоторого периода времени.

## Решение

Создайте экземпляр класса `UILongPressGestureRecognizer` и добавьте его к виду, в котором требуется распознавать жесты долгого нажатия. `.h`-файл контроллера вида будет определяться следующим образом:

```
#import "ViewController.h"

@interface ViewController ()

@property (nonatomic, strong)
  UILongPressGestureRecognizer *longPressGestureRecognizer;

@property (nonatomic, strong) UIButton *dummyButton;

@end
@implementation ViewController
```

Далее приведен метод экземпляра `viewDidLoad`, относящийся к контроллеру вида, где используется распознаватель долгих нажатий. Этот распознаватель реализован в следующем `.m`-файле:

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


  self.dummyButton = [UIButton buttonWithType:UIButtonTypeRoundedRect];
```

```
    self.dummyButton.frame = CGRectMake(0.0f,
                                        0.0f,
                                        72.0f,
                                        37.0f);
    self.dummyButton.center = self.view.center;
    [self.view addSubview:self.dummyButton];

    /* Сначала создаем распознаватель жестов. */
    self.longPressGestureRecognizer =
    [[UILongPressGestureRecognizer alloc]
      initWithTarget:self
      action:@selector(handleLongPressGestures:)];

    /* Количество пальцев, которые должны находиться на экране. */
    self.longPressGestureRecognizer.numberOfTouchesRequired = 2;

    /* Допускается движение не более чем на 100 точек,
       прежде чем жест будет распознан. */
    self.longPressGestureRecognizer.allowableMovement = 100.0f;

    /* Пользователь должен прижать к экрану два пальца
       (numberOfTouchesRequired) как минимум на секунду, чтобы жест
        был распознан. */
    self.longPressGestureRecognizer.minimumPressDuration = 1.0;

    /* Добавляем этот распознаватель жестов к виду. */
    [self.view addGestureRecognizer:self.longPressGestureRecognizer];

}
```

Если распознаватель долгих нажатий инициирует события, отправляемые объекту-получателю, а пользователь продолжает совершать такой жест и в этой ситуации поступает входящий звонок либо наступает какое-то иное прерывание, то распознаватель жестов перейдет в состояние UIGestureRecognizerStateCancelled. Объекту-получателю не будет поступать никакой информации от распознавателя жестов до тех пор, пока пользователь снова не совершит всю последовательность действий, требуемых, чтобы возобновился процесс распознавания. В данном примере распознавание возобновится после удержания хотя бы двух пальцев на виде в контроллере вида, и нажатие должно длиться не менее 1 секунды.

Код работает в контроллере вида со свойством longPressGestureRecognizer типа UILongPressGestureRecognizer. Этот аспект подробнее рассмотрен в подразделе «Решение» данного раздела.

## Обсуждение

В составе iOS SDK есть класс для распознавания долгих нажатий, который называется `UILongTapGestureRecognizer`. Жест долгого нажатия инициируется, когда пользователь нажимает одним или несколькими пальцами (количество пальцев в данном

случае задает программист) вид UIView и удерживает палец (или пальцы) в этой точке на протяжении определенного количества секунд. Учитывайте, что долгие нажатия — это непрерывные события.

На работу распознавателя жестов долгих нажатий влияют четыре важных свойства:

- numberOfTapsRequired — это количество нажатий целевого вида, которые пользователь должен совершить, прежде чем может быть инициирован жест долгого нажатия. Не забывайте, что *нажать* — это не просто *прикоснуться* пальцем к экрану. Нажатие — это движение, при котором палец сначала прижимается к экрану, а потом отрывается от него. По умолчанию это свойство имеет значение 0;
- numberOfTouchesRequired — в этом свойстве указывается количество пальцев, которые должны оказаться на экране, прежде чем начнется распознавание жеста. Если свойство numberOfTapsRequired имеет значение больше 0, то для обнаружения нажатий нужно указать аналогичное количество пальцев;
- allowableMovement — это максимальное количество пикселов, на которое можно продвинуть палец на экране, прежде чем распознавание жеста будет прекращено;
- minimumPressDuration — данное свойство указывает, как долго (в секундах) пользователь должен прижимать пальцы к экрану, прежде чем будет обнаружен жест.

В нашем примере для перечисленных свойств заданы следующие значения:

- numberOfTapsRequired — Default (это значение мы не меняем);
- numberOfTouchesRequired — 2;
- allowableMovement — 100;
- minimumPressDuration — 1.

При таких значениях жест долгого нажатия будет распознан, только если пользователь нажмет экран двумя пальцами и задержит пальцы на экране в течение 1 секунды (minimumPressDuration), причем перемещать пальцы от места касания он может не более чем на 100 пикселов (allowableMovement).

Теперь, когда жест распознан, он вызовет метод handleLongPressGestures:, который можно реализовать следующим образом:

```
- (void) handleLongPressGestures:
                        (UILongPressGestureRecognizer *)paramSender{

  /* Здесь мы хотим найти среднюю точку между двумя пальцами,
     инициировавшими жест долгого нажатия, который требуется распознать.
     Мы сконфигурировали это число, воспользовавшись свойством
     numberOfTouchesRequired класса UILongPressGestureRecognizer,
     инстанцированного в методе экземпляра viewDidLoad данного контроллера
     вида. Если выяснится, что другой распознаватель долгих нажатий
     использует данный метод в качестве целевого, мы это проигнорируем. */

    if (paramSender.numberOfTouchesRequired == 2){
      CGPoint touchPoint1 =
```

```
        [paramSender locationOfTouch:0
                              inView:paramSender.view];

        CGPoint touchPoint2 =
        [paramSender locationOfTouch:1
                              inView:paramSender.view];

        CGFloat midPointX = (touchPoint1.x + touchPoint2.x) / 2.0f;
        CGFloat midPointY = (touchPoint1.y + touchPoint2.y) / 2.0f;

        CGPoint midPoint = CGPointMake(midPointX, midPointY);
        self.dummyButton.center = midPoint;

      } else {
        /* Это распознаватель долгих нажатий, которые совершаются
           более или менее чем двумя пальцами. */

      }
    }

}
```

В качестве примера программы для iOS, в которой используются долгие нажатия, можно назвать приложение Maps (Карты). Просматривая в этой программе разные места, нажмите пальцем определенную точку на карте и ненадолго задержите палец. В этой точке появится маркер.

# 10.5. Обнаружение жестов-нажатий

## Постановка задачи

Необходимо фиксировать, когда пользователь нажимает экранный вид в той или иной точке.

## Решение

Создайте экземпляр класса `UITapGestureRecognizer` и добавьте его к целевому виду с помощью метода экземпляра `addGestureRecognizer:`, относящегося к классу `UIView`. Рассмотрим определение контроллера вида (`.h`-файл):

```
#import "ViewController.h"

@interface ViewController ()

@property (nonatomic, strong)
  UITapGestureRecognizer *tapGestureRecognizer;

@end
@implementation ViewController
```

Реализация метода экземпляра viewDidLoad контроллера вида такова:

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

  /* Создаем распознаватель жестов-нажатий. */
  self.tapGestureRecognizer = [[UITapGestureRecognizer alloc]
                               initWithTarget:self
                               action:@selector(handleTaps:)];

  /* Количество пальцев, которые должны находиться на экране. */
  self.tapGestureRecognizer.numberOfTouchesRequired = 2;

  /* Общее количество касаний, которое должно быть выполнено, прежде
     чем жест будет распознан. */
  self.tapGestureRecognizer.numberOfTapsRequired = 3;

  /* Добавляем к виду этот распознаватель жестов. */
  [self.view addGestureRecognizer:self.tapGestureRecognizer];

}
```

## Обсуждение

Распознаватель жестов-нажатий лучше всех остальных распознавателей подходит для обнаружения обычных нажатий (толчков) на экран. Нажатие — это событие, происходящее, когда пользователь касается пальцем какой-то точки экрана, а потом отрывает палец от него. Жест нажатия является дискретным.

Метод locationInView: класса UITapGestureRecognizer можно применять для обнаружения точки, в которой произошло событие нажатия. Если для регистрации нажатия требуется более одного касания, то можно вызвать метод locationOfTouch:inView: класса UITapGestureRecognizer, определяющий точки отдельных касаний. В коде мы задали для свойства numberOfTouchesRequired распознавателя жестов-нажатий значение 2. При таком значении распознавателю жестов необходимо, чтобы в момент каждого нажатия на экране находились два пальца. Количество нажатий, требуемое, чтобы жесты-нажатия стали распознаваться, определено как 3. Я сделал это с помощью свойства numberOfTapsRequired. Я указал метод handleTaps: в качестве целевого метода распознавателя жестов-нажатий:

```
- (void) handleTaps:(UITapGestureRecognizer*)paramSender{

  NSUInteger touchCounter = 0;
  for (touchCounter = 0;
       touchCounter < paramSender.numberOfTouchesRequired;
       touchCounter++){
    CGPoint touchPoint =
    [paramSender locationOfTouch:touchCounter
                          inView:paramSender.view];
    NSLog(@"Touch #%lu: %@",
```

```
            (unsigned long)touchCounter+1,
            NSStringFromCGPoint(touchPoint));
    }

}
```

В этом коде мы дожидаемся, пока произойдет такое количество нажатий, которое требуется, чтобы распознаватель жестов-нажатий начал работу. Беря за основу это количество, мы узнаем точку, в которой было сделано каждое нажатие. В зависимости от того, работаете ли вы с реальным устройством или с симулятором, в окне консоли будут выведены примерно такие результаты:

```
Touch #1: {107, 186}
Touch #2: {213, 254}
```

При работе с симулятором можно имитировать два одновременных нажатия, удерживая клавишу Option и передвигая указатель мыши по экрану симулятора. На экране появятся две концентрические точки касания.

Кроме того, необходимо упомянуть о методе `NSStringFromCGPoint`, который, как понятно из его названия[1], может преобразовать структуру `CGPoint` в строку `NSString`. Эта функция применяется для превращения `CGPoint` каждого прикосновения к экрану в `NSString`, а данную строку мы уже можем записать в окне консоли, воспользовавшись `NSLog`. Чтобы открыть окно консоли, выполните **Run ▸ Console** (Запустить ▸ Консоль).

# 10.6. Обнаружение щипка

## Постановка задачи

Необходимо предоставить пользователю возможность выполнять движения щипка.

## Решение

Создайте экземпляр класса `UIPinchGestureRecognizer` и добавьте его к вашему целевому виду, воспользовавшись методом экземпляра `addGestureRecognizer:`, относящимся к классу `UIView`:

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

  CGRect labelRect = CGRectMake(0.0f,     /* X */
                                0.0f,     /* Y */
                                200.0f,   /* Ширина */
                                200.0f);  /* Высота */
```

[1] На преобразование указывает компонент from из названия метода. — *Примеч. пер.*

```
  self.myBlackLabel = [[UILabel alloc] initWithFrame:labelRect];
  self.myBlackLabel.center = self.view.center;
  self.myBlackLabel.backgroundColor = [UIColor blackColor];

  /* Без этой строки распознаватель щипков работать не будет. */
  self.myBlackLabel.userInteractionEnabled = YES;
  [self.view addSubview:self.myBlackLabel];

  /* Создаем распознаватель щипков. */
  self.pinchGestureRecognizer =  [[UIPinchGestureRecognizer alloc]
                                  initWithTarget:self
                                  action:@selector(handlePinches:)];

  /* Добавляем этот распознаватель жестов к виду. */
  [self.myBlackLabel
    addGestureRecognizer:self.pinchGestureRecognizer];

}
```

Контроллер вида определяется так:

```
#import "ViewController.h"

@interface ViewController ()
@property (nonatomic, strong)
  UIPinchGestureRecognizer *pinchGestureRecognizer;

@property (nonatomic, strong) UILabel *myBlackLabel;

@property (nonatomic, unsafe_unretained) CGFloat currentScale;

@end
```

## Обсуждение

Щипки позволяют пользователю легко масштабировать (увеличивать и уменьшать) элементы графического интерфейса. Например, браузер Safari в iOS дает возможность щипком на веб-странице увеличивать ее содержимое. Щипок работает в двух направлениях: увеличение и уменьшение масштаба. Это непрерывный жест, который на сенсорном экране всегда выполняется двумя пальцами.

Данный распознаватель жестов может пребывать в следующих состояниях:

- `UIGestureRecognizerStateBegan`;
- `UIGestureRecognizerStateChanged`;
- `UIGestureRecognizerStateEnded`.

Как только щипок распознан, вызывается действующий метод целевого объекта (и будет последовательно вызываться до тех пор, пока щипок не окончится). В действующем методе вы получаете доступ к двум очень важным методам распознавателя щипков: `scale` и `velocity`. `scale` — это коэффициент, на который нужно

изменить размер элемента графического интерфейса по осям *X* и *Y*, чтобы отразить таким образом размер пользовательского жеста. `velocity` — это скорость щипка, измеряемая в пикселах в секунду. Скорость имеет отрицательное значение, если пальцы движутся навстречу друг другу, и положительное — если они перемещаются в разные стороны.

Значение свойства `scale` можно передать функции `CGAffineTransformMakeScale` из фреймворка Core Graphics, чтобы получить аффинное преобразование. Такое преобразование применимо к свойству `transform` любого экземпляра класса `UIView`, оно позволяет изменять преобразование этого элемента. Мы воспользуемся этой функцией следующим образом:

```
- (void) handlePinches:(UIPinchGestureRecognizer*)paramSender{

  if (paramSender.state == UIGestureRecognizerStateEnded){
    self.currentScale = paramSender.scale;
  } else if (paramSender.state == UIGestureRecognizerStateBegan &&
             self.currentScale != 0.0f){
    paramSender.scale = self.currentScale;
  }

  if (paramSender.scale != NAN &&
      paramSender.scale != 0.0){
    paramSender.view.transform =
    CGAffineTransformMakeScale(paramSender.scale,
                               paramSender.scale);
  }

}
```

Поскольку свойство `scale` распознавателя жестов сбрасывается всякий раз, когда регистрируется новый щипок, мы сохраняем последнее значение этого свойства в общем свойстве экземпляра (Instance Property) контроллера вида, называемом `currentScale`. В следующий раз, когда будет распознан новый жест, мы отсчитываем коэффициент масштабирования от последнего зафиксированного значения, что и продемонстрировано в коде.

# 11 Сетевые функции, JSON, XML и Twitter

## 11.0. Введение

Стоит подключить приложение iOS к Интернету — и оно становится гораздо интереснее. Например, представьте себе приложение, которое предлагает пользователям великолепные фоновые картинки для Рабочего стола. Пользователь может выбрать вариант из большого списка изображений и присвоить любой из этих рисунков в качестве фонового операционной системе iOS. А теперь вообразим себе приложение, которое делает то же самое, но обновляет ассортимент имеющихся изображений каждый день, неделю или месяц. Пользователь после какого-то перерыва возвращается к работе с программой и — опа! Масса новых фоновых изображений динамически загружается в приложение. В этом и есть изюминка работы с веб-службами и Интернетом. Реализовать такие функции не составляет труда, если обладать базовыми знаниями о работе в Сети, применении JSON, XML и Twitter. Ну, еще от разработчика приложения требуется известная креативность.

iOS SDK позволяет подключаться к Интернету, получать и отсылать данные. Это делается с помощью класса `NSURLConnection`. Сериализация и десериализация JSON выполняется в классе `NSJSONSerialization`. Синтаксический разбор XML производится с помощью `NSXMLParser`, а соединение с Twitter обеспечивается во фреймворке Twitter.

В SDK iOS 7 появились новые классы, работать с которыми мы научимся в этой главе. В частности, поговорим о классе `NSURLSession`, который управляет соединяемостью веб-сервисов и решает эту задачу более основательно, чем класс `NSURLConnection`. О соединяемости мы также поговорим далее в этой главе.

## 11.1. Асинхронная загрузка с применением NSURLConnection

### Постановка задачи

Необходимо асинхронно загрузить файл с имеющегося URL.

## Решение

Используйте класс `NSURLConnection` с асинхронным запросом.

## Обсуждение

Класс `NSURLConnection` можно использовать двумя способами — асинхронным и синхронным. При асинхронном соединении создается новый поток, и процесс загрузки выполняется в этом новом потоке. Синхронное соединение блокирует *вызывающий поток*, а содержимое загружается прямо в ходе обмена данными.

Многие разработчики полагают, что при синхронном соединении блокируется главный *поток*, но это неверно. Синхронное соединение всегда блокирует тот поток, в котором оно было инициировано. Если вы запускаете синхронное соединение из главного потока — да, главный поток будет заблокирован. Но синхронное соединение, запущенное из другого потока, будет напоминать асинхронное именно в том отношении, что оно никак не повлияет на главный поток. На самом деле единственное различие между синхронным и асинхронным соединениями заключается в том, что для асинхронного соединения среда времени исполнения создает отдельный поток, а для синхронного — нет.

Чтобы создать асинхронное соединение, необходимо следующее.

1. Иметь URL или экземпляр `NSString`.
2. Преобразовать строку в экземпляр `NSURL`.
3. Поместить URL в URL-запросе типа `NSURLRequest`, а если мы имеем дело с изменяемыми URL — в экземпляр `NSMutableURLRequest`.
4. Создать экземпляр `NSURLConnection` и передать ему URL-запрос.

Можно создать асинхронное соединение по URL с помощью метода класса `sendAsynchronousRequest:queue:completionHandler:`, относящегося к классу `NSURLConnection`. Этот метод имеет следующие параметры:

- `sendAsynchronousRequest` — запрос типа `NSURLRequest`, рассмотренный ранее;
- `queue` — операционная очередь. При желании можно просто выделить и инициализировать новую операционную очередь и передать ее этому методу;
- `completionHandler` — блоковый объект, выполняемый, когда асинхронное соединение завершает работу, успешно или неуспешно. Этот блоковый объект должен принимать три параметра:
  - объект типа `NSURLResponse`, в котором заключается ответ, полученный нами от сервера, — при наличии такого ответа;
  - данные типа `NSData` при их наличии. Это будут данные, собранные в ходе соединения по указанному URL;
  - ошибка типа `NSError` в случае ее возникновения.

Метод sendAsynchronousRequest:queue:completionHandler: не вызывается в главном потоке. Поэтому, если вам потребуется решить задачу, связанную с пользовательским интерфейсом, убедитесь, что вернулись к главному потоку.

Итак, довольно теории, перейдем к примерам. В данном примере попытаемся собрать HTML-контент с домашней страницы Apple, а потом выведем эту информацию в строковом формате в окне консоли:

```
NSString *urlAsString = @"http://www.apple.com";
NSURL *url = [NSURL URLWithString:urlAsString];
NSURLRequest *urlRequest = [NSURLRequest requestWithURL:url];
NSOperationQueue *queue = [[NSOperationQueue alloc] init];

[NSURLConnection
 sendAsynchronousRequest:urlRequest
 queue:queue
 completionHandler:^(NSURLResponse *response,
                     NSData *data,
                     NSError *error) {

   if ([data length] >0  &&
       error == nil){
     NSString *html = [[NSString alloc] initWithData:data
                                            encoding:NSUTF8StringEncoding];
     NSLog(@"HTML = %@", html);
   }
   else if ([data length] == 0 &&
            error == nil){
     NSLog(@"Nothing was downloaded.");
   }
   else if (error != nil){
     NSLog(@"Error happened = %@", error);
   }

}];
```

Да, все так просто. Если вы хотите сохранить данные, которые мы загрузили на диск в ходе соединения, это можно сделать с помощью подходящих методов класса NSData, получаемых от завершающего блока:

```
NSString *urlAsString = @"http://www.apple.com";
NSURL *url = [NSURL URLWithString:urlAsString];
NSURLRequest *urlRequest = [NSURLRequest requestWithURL:url];
NSOperationQueue *queue = [[NSOperationQueue alloc] init];

  [NSURLConnection
   sendAsynchronousRequest:urlRequest
   queue:queue
   completionHandler:^(NSURLResponse *response,
                       NSData *data,
                       NSError *error) {

     if ([data length] >0  &&
         error == nil){
```

```
        /* Прикрепляем имя файла к каталогу с документами. */

        NSURL *filePath =
        [[self documentsFolderUrl]
        URLByAppendingPathComponent:@"apple.html"];

        [data writeToURL:filePath atomically:YES];

        NSLog(@"Successfully saved the file to %@", filePath);

      }
      else if ([data length] == 0 &&
               error == nil){
        NSLog(@"Nothing was downloaded.");
      }
      else if (error != nil){
        NSLog(@"Error happened = %@", error);
      }

}];
```

Все действительно просто. В более ранних версиях iOS SDK соединения по URL происходили с применением делегирования, но теперь модель стала обычной блоковой и вам не придется заниматься реализацией делегатов.

# 11.2. Обработка задержек при асинхронных соединениях

Необходимо задать лимит ожидания — проще говоря, задержку — при асинхронном соединении.

## Решение

Задайте задержку в URL-запросе, посылаемом классу `NSURLConnection`.

## Обсуждение

При инстанцировании объекта типа `NSURLRequest` для передачи URL-соединения можно воспользоваться методом класса `requestWithURL:cachePolicy:timeoutInterval:`, относящимся к этому объекту, и передать желаемую длительность задержки в секундах в параметре `timeoutInterval`.

Например, если вы готовы не более 30 секунд дожидаться, пока загрузится содержимое главной страницы Apple (с применением синхронного соединения), создайте ваш URL таким образом:

```
- (BOOL) application:(UIApplication *)application
  didFinishLaunchingWithOptions:(NSDictionary *)launchOptions{
    NSString *urlAsString = @"http://www.apple.com";
    NSURL *url = [NSURL URLWithString:urlAsString];

    NSURLRequest *urlRequest =
    [NSURLRequest
      requestWithURL:url
      cachePolicy:NSURLRequestReloadIgnoringLocalAndRemoteCacheData
      timeoutInterval:30.0f];

    NSOperationQueue *queue = [[NSOperationQueue alloc] init];

    [NSURLConnection
      sendAsynchronousRequest:urlRequest
      queue:queue
      completionHandler:^(NSURLResponse *response,
                          NSData *data,
                          NSError *error) {

        if ([data length] >0  &&
          error == nil){
          NSString *html = [[NSString alloc] initWithData:data
                                                 encoding:NSUTF8StringEncoding];
          NSLog(@"HTML = %@", html);
        }
        else if ([data length] == 0 &&
          error == nil){
          NSLog(@"Nothing was downloaded.");
        }
        else if (error != nil){
        NSLog(@"Error happened = %@", error);
      }

    }];
    self.window = [[UIWindow alloc]
                   initWithFrame:[[UIScreen mainScreen] bounds]];
    self.window.backgroundColor = [UIColor whiteColor];
    [self.window makeKeyAndVisible];
    return YES;
}
```

Что же здесь происходит? Дело в том, что среда времени исполнения пытается получить содержимое, расположенное по предоставленной ссылке. Если это удается сделать в течение заданных 30 секунд и соединение устанавливается до возникновения задержки — хорошо. В противном случае среда времени исполнения выдаст вам ошибку задержки (`error`) в соответствующем параметре завершающего блока.

# 11.3. Синхронная загрузка с применением NSURLConnection

## Постановка задачи

Необходимо синхронно загрузить информацию, расположенную по имеющемуся URL.

## Решение

Используйте метод класса `sendSynchronousRequest:returningResponse:error:`, относящийся к классу `NSURLConnection`. Возвращаемое значение этого метода — данные типа `NSData`.

## Обсуждение

Пользуясь методом класса `sendSynchronousRequest:returningResponse:error:`, относящимся к классу `NSURLConnection`, можно посылать синхронный запрос к URL. А теперь внимание! Синхронные соединения *не обязательно* блокируют главный поток. Эти соединения блокируют *актуальный поток*, то есть выполняющий текущую задачу, и если этот поток не главный, то главный поток останется свободным. Если приступить к обработке глобальной параллельной очереди в GCD, а потом инициировать синхронное соединение, то вы *не заблокируете* главный поток.

Попробуем инициировать наше первое синхронное соединение и посмотрим, что произойдет. В данном примере мы попытаемся получить домашнюю страницу сайта Yahoo!:

```
- (BOOL)            application:(UIApplication *)application
  didFinishLaunchingWithOptions:(NSDictionary *)launchOptions{

    NSLog(@"We are here...");

    NSString *urlAsString = @"http://www.yahoo.com";
    NSURL *url = [NSURL URLWithString:urlAsString];

    NSURLRequest *urlRequest = [NSURLRequest requestWithURL:url];

    NSURLResponse *response = nil;
    NSError *error = nil;

    NSLog(@"Firing synchronous url connection...");
    NSData *data = [NSURLConnection sendSynchronousRequest:urlRequest
                                         returningResponse:&response
                                                     error:&error];
```

```
    if ([data length] > 0 &&
        error == nil){
      NSLog(@"%lu bytes of data was returned.", (unsigned long)[data length]);
    }
    else if ([data length] == 0 &&
             error == nil){
      NSLog(@"No data was returned.");
    }
    else if (error != nil){
      NSLog(@"Error happened = %@", error);
    }

    NSLog(@"We are done.");

    self.window = [[UIWindow alloc] initWithFrame:
                   [[UIScreen mainScreen] bounds]];

    self.window.backgroundColor = [UIColor whiteColor];
    [self.window makeKeyAndVisible];
    return YES;
}
```

Если запустить это приложение, а потом взглянуть в окно консоли, то там окажется выведен следующий результат:

```
We are here...
Firing synchronous url connection...
252117 bytes of data was returned.
We are done.
```

Итак, вполне очевидно, что актуальный поток написал на консоли строку `We are here...`, дождался окончания соединения (поскольку это синхронное соединение, блокирующее актуальный поток), а потом вывел в окне консоли текст `We are done`. Теперь проведем эксперимент. Поместим то же самое синхронное соединение в глобальной параллельной очереди в GCD, то есть гарантированно обеспечим параллелизм, и посмотрим, что произойдет:

```
- (BOOL)            application:(UIApplication *)application
  didFinishLaunchingWithOptions:(NSDictionary *)launchOptions{

    NSLog(@"We are here...");

    NSString *urlAsString = @"http://www.yahoo.com";

    NSLog(@"Firing synchronous url connection...");

    dispatch_queue_t dispatchQueue =
      dispatch_get_global_queue(DISPATCH_QUEUE_PRIORITY_DEFAULT, 0);

    dispatch_async(dispatchQueue, ^(void) {

      NSURL *url = [NSURL URLWithString:urlAsString];
```

```
    NSURLRequest *urlRequest = [NSURLRequest requestWithURL:url];
    NSURLResponse *response = nil;
    NSError *error = nil;

    NSData *data = [NSURLConnection sendSynchronousRequest:urlRequest
                                         returningResponse:&response
                                                     error:&error];

    if ([data length] > 0 &&
        error == nil){
      NSLog(@"%lu bytes of data was returned.", (unsigned long)[data length]);
    }
    else if ([data length] == 0 &&
             error == nil){
      NSLog(@"No data was returned.");
    }
    else if (error != nil){
      NSLog(@"Error happened = %@", error);
    }
  });

  NSLog(@"We are done.");

  self.window = [[UIWindow alloc] initWithFrame:
                 [[UIScreen mainScreen] bounds]];
  self.window.backgroundColor = [UIColor whiteColor];
  [self.window makeKeyAndVisible];
  return YES;
}
```

Вывод будет примерно таким:

```
We are here...
Firing synchronous url connection...
We are done.
252450 bytes of data was returned.
```

Итак, в данном примере текущий поток вывел текст `We are done` в окне консоли, не дожидаясь, пока синхронное соединение завершит считывание с заданного URL. Интересно, правда? Таким образом, этот пример доказывает, что при умелом обращении синхронное URL-соединение не обязательно блокирует главный поток. Тем не менее оно гарантированно блокирует *текущий* поток.

# 11.4. Изменение URL-запроса с применением NSMutableURLRequest

## Постановка задачи

Требуется корректировать различные HTTP-заголовки и настройки URL-запроса перед передачей его URL-соединению.

## Решение

Эта техника лежит в основе некоторых разделов, рассмотренных далее в этой главе. Пользуйтесь `NSMutableURLRequest` вместо `NSURLRequest`.

## Обсуждение

URL-запрос может быть *изменяемым* или *неизменяемым*. URL-запросы, относящиеся к первой категории, поддаются изменениям после выделения и инициализации, а те, что относятся ко второй категории, — нет. Этот раздел посвящен изменяемым URL-запросам. Их можно создавать с помощью класса `NSMutableURLRequest`.

Рассмотрим пример, в котором длительность задержки при URL-запросе изменяется *после* выделения и инициализации этого запроса:

```
NSString *urlAsString = @"http://www.apple.com";
NSURL *url = [NSURL URLWithString:urlAsString];

NSMutableURLRequest *urlRequest = [NSMutableURLRequest requestWithURL:url];

[urlRequest setTimeoutInterval:30.0f];
```

Теперь обратимся к другому примеру, где URL и время задержки при URL-запросе задаются после выделения и инициализации:

```
NSString *urlAsString = @"http://www.apple.com";
NSURL *url = [NSURL URLWithString:urlAsString];
NSMutableURLRequest *urlRequest = [NSMutableURLRequest new];
[urlRequest setTimeoutInterval:30.0f];
[urlRequest setURL:url];
```

В других разделах этой главы мы изучим некоторые очень тонкие приемы, которые осуществимы с помощью изменяемых URL-запросов.

# 11.5. Отправка запросов HTTP GET с применением NSURLConnection

## Постановка задачи

Необходимо отправить запрос `GET` по протоколу HTTP и, возможно, передать получателю вместе с этим запросом какие-либо параметры.

## Решение

По определению `GET`-запросы допускают указание параметров в строках запросов в общеизвестной форме:

```
http://example.com/?param1=value1&param2=value2...
```

Строки можно использовать для перечисления параметров в обычном формате.

## Обсуждение

GET-запрос — это запрос к веб-серверу на получение данных. Обычно запрос сопровождается параметрами, которые отправляются в строке запроса как часть URL.

Чтобы протестировать вызов GET, необходимо найти веб-сервер, принимающий такие вызовы и способный отослать какие-либо данные в ответ. Это просто. Как вы уже знаете, при открытии веб-страницы в браузере этот браузер по умолчанию посылает запрос GET к конечной точке. Поэтому данный раздел вы можете опробовать на любом сайте по своему усмотрению.

Для симулирования отправки параметров строки запроса в GET-запросе к той же веб-службе с помощью NSURLConnection воспользуемся изменяемым URL-запросом и явно укажем ваш HTTP-метод для GET с помощью метода setHTTPMethod:, относящегося к NSMutableURLRequest. Параметры оформляются как часть URL, следующим образом:

```
- (BOOL) application:(UIApplication *)application
didFinishLaunchingWithOptions:(NSDictionary *)launchOptions{

  NSString *urlAsString = <# Здесь укажите URL веб-сервера #>;

  urlAsString = [urlAsString stringByAppendingString:@"?param1=First"];
  urlAsString = [urlAsString stringByAppendingString:@"&param2=Second"];

  NSURL *url = [NSURL URLWithString:urlAsString];

  NSMutableURLRequest *urlRequest = [NSMutableURLRequest
                                     requestWithURL:url];
  [urlRequest setTimeoutInterval:30.0f];
  [urlRequest setHTTPMethod:@"GET"];

  NSOperationQueue *queue = [[NSOperationQueue alloc] init];

  [NSURLConnection
   sendAsynchronousRequest:urlRequest
   queue:queue
   completionHandler:^(NSURLResponse *response,
                       NSData *data,
                       NSError *error) {
    if ([data length] >0  &&
        error == nil){
      NSString *html = [[NSString alloc] initWithData:data
                                        encoding:NSUTF8StringEncoding];
      NSLog(@"HTML = %@", html);
    }
    else if ([data length] == 0 &&
             error == nil){
      NSLog(@"Nothing was downloaded.");
    }
    else if (error != nil){
```

```
        NSLog(@"Error happened = %@", error);
      }

    }];
    self.window = [[UIWindow alloc] initWithFrame:
                   [[UIScreen mainScreen] bounds]];

    self.window.backgroundColor = [UIColor whiteColor];
    [self.window makeKeyAndVisible];
    return YES;
}
```

Переменная urlAsString в данном коде представляет собой сущность Xcode, которая называется «шаблон переменной». Если скопировать этот код и вставить его в ваш проект Xcode, переменная будет отображена так, как показано на рис. 11.1. Перед запуском этого примера кода убедитесь, что присвоили вышеупомянутой переменной валидный URL.

```
NSString *urlAsString = Place the URL of the web server here;
urlAsString = [urlAsString stringByAppendingString:@"?param1=First"];
urlAsString = [urlAsString stringByAppendingString:@"&param2=Second"];
```

**Рис. 11.1.** Заменяемая переменная в Xcode

Единственный момент, который необходимо учитывать, заключается в том, что перед первым параметром ставится вопросительный знак, а перед всеми последующими — амперсанд (`&`). Вот и все! Теперь вы можете пользоваться методом HTTP `GET` и отправлять параметры в строке запроса.

# 11.6. Отправка запросов HTTP POST с применением NSURLConnection

## Постановка задачи

Необходимо вызвать метод HTTP `POST` веб-сервера и, возможно, передать параметры (в теле HTTP или в строке запроса) определенной веб-службе.

## Решение

Как и в случае с методом `GET`, можно использовать метод `POST` с применением `NSURLConnection`. Следует явно задать метод нашего URL как `POST`.

## Обсуждение

Напишем простое приложение, которое может создать асинхронное соединение и отослать ряд параметров в виде строки запроса и нескольких параметров в теле HTTP-запроса по URL:

```
- (BOOL) application:(UIApplication *)application
didFinishLaunchingWithOptions:(NSDictionary *)launchOptions{
  NSString *urlAsString = <# Здесь укажите URL веб-сервера #>;

  urlAsString = [urlAsString stringByAppendingString:@"?param1=First"];
  urlAsString = [urlAsString stringByAppendingString:@"&param2=Second"];

  NSURL *url = [NSURL URLWithString:urlAsString];

  NSMutableURLRequest *urlRequest = [NSMutableURLRequest requestWithURL:url];
  [urlRequest setTimeoutInterval:30.0f];
  [urlRequest setHTTPMethod:@"POST"];

  NSString *body = @"bodyParam1=BodyValue1&bodyParam2=BodyValue2";
  [urlRequest setHTTPBody:[body dataUsingEncoding:NSUTF8StringEncoding]];

  NSOperationQueue *queue = [[NSOperationQueue alloc] init];
  [NSURLConnection
   sendAsynchronousRequest:urlRequest
   queue:queue
   completionHandler:^(NSURLResponse *response,
                       NSData *data,
                       NSError *error) {

    if ([data length] >0  &&
        error == nil){
      NSString *html = [[NSString alloc] initWithData:data
                                             encoding:NSUTF8StringEncoding];
      NSLog(@"HTML = %@", html);
    }
    else if ([data length] == 0 &&
             error == nil){
      NSLog(@"Nothing was downloaded.");
    }
    else if (error != nil){
      NSLog(@"Error happened = %@", error);
    }

  }];
  self.window = [[UIWindow alloc] initWithFrame:
                 [[UIScreen mainScreen] bounds]];

  self.window.backgroundColor = [UIColor whiteColor];
  [self.window makeKeyAndVisible];
  return YES;
}
```

Первый параметр, пересылаемый в теле HTTP, не обязательно предварять вопросительным знаком, а пересылаемый в строке запроса — обязательно.

# 11.7. Отправка запросов HTTP DELETE с применением NSURLConnection

## Постановка задачи

Требуется вызвать веб-службу методом HTTP DELETE, чтобы удалить ресурс, расположенный по ссылке URL, и, возможно, передать веб-службе определенные параметры, которые будут находиться в теле HTTP или в строке запроса.

## Решение

Как и методы GET и POST, метод DELETE можно использовать с помощью NSURLConnection. Необходимо явно задать метод вашего URL как DELETE.

## Обсуждение

Напишем простое приложение, которое будет создавать асинхронное соединение и отправлять несколько параметров в строке запроса, а несколько — в теле HTTP. Отправка будет происходить по указанному URL с помощью метода DELETE HTTP:

```
- (BOOL) application:(UIApplication *)application
didFinishLaunchingWithOptions:(NSDictionary *)launchOptions{
  NSString *urlAsString = <# Здесь укажите URL веб-сервера #>;

  urlAsString = [urlAsString stringByAppendingString:@"?param1=First"];
  urlAsString = [urlAsString stringByAppendingString:@"&param2=Second"];

  NSURL *url = [NSURL URLWithString:urlAsString];

  NSMutableURLRequest *urlRequest = [NSMutableURLRequest requestWithURL:url];
  [urlRequest setTimeoutInterval:30.0f];
  [urlRequest setHTTPMethod:@"DELETE"];

  NSString *body = @"bodyParam1=BodyValue1&bodyParam2=BodyValue2";
  [urlRequest setHTTPBody:[body dataUsingEncoding:NSUTF8StringEncoding]];

  NSOperationQueue *queue = [[NSOperationQueue alloc] init];

  [NSURLConnection
   sendAsynchronousRequest:urlRequest
   queue:queue
   completionHandler:^(NSURLResponse *response,
                       NSData *data,
                       NSError *error) {

    if ([data length] >0  &&
        error == nil){
      NSString *html = [[NSString alloc] initWithData:data
```

```
                                                encoding:NSUTF8StringEncoding];
      NSLog(@"HTML = %@", html);
    }
    else if ([data length] == 0 &&
             error == nil){
      NSLog(@"Nothing was downloaded.");
    }
    else if (error != nil){
      NSLog(@"Error happened = %@", error);
    }

  }];
  self.window = [[UIWindow alloc] initWithFrame:
                 [[UIScreen mainScreen] bounds]];

  self.window.backgroundColor = [UIColor whiteColor];
  [self.window makeKeyAndVisible];
  return YES;
}
```

Этот пример очень напоминает код, рассмотренный в разделах 11.5 и 11.6. Разница заключается в том, что здесь мы использовали HTTP-метод `DELETE`. Прочая информация практически идентична той, что была изложена в упомянутых разделах.

# 11.8. Отправка запросов HTTP PUT с применением NSURLConnection

## Постановка задачи

Требуется вызывать веб-службу методом HTTP `PUT`, чтобы размещать ресурс на веб-сервере и, возможно, передать веб-службе определенные параметры, которые будут находиться в теле HTTP или в строке запроса.

## Решение

Как и методы `GET`, `POST` и `DELETE`, метод `PUT` можно использовать с помощью `NSURLConnection`. Необходимо явно задать метод вашего URL как `PUT`.

## Обсуждение

Напишем простое приложение, которое будет создавать асинхронное соединение и отправлять несколько параметров в строке запроса, а несколько — в теле HTTP. Отправка будет происходить по указанному URL с помощью метода `PUT`:

```
- (BOOL) application:(UIApplication *)application
didFinishLaunchingWithOptions:(NSDictionary *)launchOptions{
```

```
    NSString *urlAsString = <# Здесь укажите URL веб-сервера #>;

    urlAsString = [urlAsString stringByAppendingString:@"?param1=First"];
    urlAsString = [urlAsString stringByAppendingString:@"&param2=Second"];

    NSURL *url = [NSURL URLWithString:urlAsString];

    NSMutableURLRequest *urlRequest = [NSMutableURLRequest requestWithURL:url];
    [urlRequest setTimeoutInterval:30.0f];
    [urlRequest setHTTPMethod:@"PUT"];

    NSString *body = @"bodyParam1=BodyValue1&bodyParam2=BodyValue2";
    [urlRequest setHTTPBody:[body dataUsingEncoding:NSUTF8StringEncoding]];

    NSOperationQueue *queue = [[NSOperationQueue alloc] init];

    [NSURLConnection
     sendAsynchronousRequest:urlRequest
     queue:queue
     completionHandler:^(NSURLResponse *response,
                         NSData *data,
                         NSError *error) {

      if ([data length] >0  &&
          error == nil){
        NSString *html = [[NSString alloc] initWithData:data
                                               encoding:NSUTF8StringEncoding];
        NSLog(@"HTML = %@", html);
      }
      else if ([data length] == 0 &&
               error == nil){
        NSLog(@"Nothing was downloaded.");
      }
      else if (error != nil){
        NSLog(@"Error happened = %@", error);
      }

    }];
    self.window = [[UIWindow alloc] initWithFrame:
                   [[UIScreen mainScreen] bounds]];

    self.window.backgroundColor = [UIColor whiteColor];
    [self.window makeKeyAndVisible];
    return YES;
}
```

Первый параметр, пересылаемый в теле HTTP, не обязательно предварять вопросительным знаком, а пересылаемый в строке запроса — обязательно.

# 11.9. Сериализация массивов и словарей в JSON

## Постановка задачи

Необходимо сериализовать словарь или массив в объект JSON, который можно передавать по сети или просто сохранять на диск.

## Решение

Воспользуйтесь методом `dataWithJSONObject:options:error:` класса `NSJSONSerialization`.

## Обсуждение

Метод `dataWithJSONObject:options:error:` класса `NSJSONSerialization` может сериализовывать словари и массивы, в которых содержатся лишь экземпляры переменных `NSString`, `NSNumber`, `NSArray`, `NSDictionary` либо `NSNull` для нулевых значений. Как было указано ранее, объект, передаваемый этому методу, должен быть либо массивом, либо словарем.

Теперь создадим простой массив с несколькими ключами и значениями:

```
NSDictionary *dictionary =
@{
  @"First Name" : @"Anthony",
  @"Last Name" : @"Robbins",
  @"Age" : @51,
  @"children" : @[
      @"Anthony's Son 1",
      @"Anthony's Daughter 1",
      @"Anthony's Son 2",
      @"Anthony's Son 3",
      @"Anthony's Daughter 2"
      ],
};
```

Как видите, в этом словаре содержатся имя, фамилия и возраст Энтони Роббинса. Ключ словаря, называемый `children`, содержит имена детей Энтони. Это массив строк, где каждой строкой представлен один ребенок. Итак, на данный момент переменная `dictionary` содержит все значения, которые мы хотели в нее поместить. Теперь нужно сериализовать ее в объект JSON:

```
NSError *error = nil;
NSData *jsonData = [NSJSONSerialization
                    dataWithJSONObject:dictionary
                    options:NSJSONWritingPrettyPrinted
                    error:&error];

if ([jsonData length] > 0 &&
```

```
    error == nil){

  NSLog(@"Successfully serialized the dictionary into data = %@", jsonData);

}
else if ([jsonData length] == 0 &&
         error == nil){

  NSLog(@"No data was returned after serialization.");

}
else if (error != nil){

  NSLog(@"An error happened = %@", error);

}
```

Возвращаемым значением метода `dataWithJSONObject:options:error:` являются данные типа `NSData`. Правда, эти данные можно просто преобразовать в строку и вывести на консоль. Для этого применяется метод-инициализатор `initWithData:encoding:` класса `NSString`. Далее приведен полный пример, в котором словарь преобразуется в объект JSON. Этот объект превращается в строку, а строка выводится в окне консоли:

```
- (BOOL) application:(UIApplication *)application
didFinishLaunchingWithOptions:(NSDictionary *)launchOptions{

  NSDictionary *dictionary =
  @{
    @"First Name" : @"Anthony",
    @"Last Name" : @"Robbins",
    @"Age" : @51,
    @"children" : @[
        @"Anthony's Son 1",
        @"Anthony's Daughter 1",
        @"Anthony's Son 2",
        @"Anthony's Son 3",
        @"Anthony's Daughter 2"
        ],
  };

  NSError *error = nil;
  NSData *jsonData = [NSJSONSerialization
                      dataWithJSONObject:dictionary
                      options:NSJSONWritingPrettyPrinted
                      error:&error];

  if ([jsonData length] > 0 &&
      error == nil){

    NSLog(@"Successfully serialized the dictionary into data.");
    NSString *jsonString = [[NSString alloc] initWithData:jsonData
```

```
                                                encoding:NSUTF8StringEncoding];
    NSLog(@"JSON String = %@", jsonString);

  }
  else if ([jsonData length] == 0 &&
           error == nil){

    NSLog(@"No data was returned after serialization.");

  }
  else if (error != nil){

    NSLog(@"An error happened = %@", error);
  self.window = [[UIWindow alloc]
                 initWithFrame:[[UIScreen mainScreen] bounds]];
  // Точка переопределения для дополнительной настройки после запуска приложения
  self.window.backgroundColor = [UIColor whiteColor];
  [self.window makeKeyAndVisible];
  return YES;
}
```

Запустив это приложение, вы увидите в окне консоли следующие результаты:

```
Successfully serialized the dictionary into data.
JSON String = {
  "Last Name" : "Robbins",
  "First Name" : "Anthony",
  "children" : [
      "Anthony's Son 1",
      "Anthony's Daughter 1",
      "Anthony's Son 2",
      "Anthony's Son 3",
      "Anthony's Daughter 2"
      ],
  "Age" : 51
}
```

# 11.10. Десериализация нотации JSON в массивы и словари

## Постановка задачи

Имеются данные в формате JSON, их необходимо десериализовать в словарь или массив.

## Решение

Воспользуйтесь методом `JSONObjectWithData:options:error:` класса `NSJSONSerialization`.

## Обсуждение

Если вы уже сериализовали ваш словарь или массив в объект JSON (заключенный в экземпляре NSData, см. раздел 11.9), то эти данные нужно будет десериализовать обратно в словарь или массив. Это делается с помощью метода JSONObjectWithData:options:error:, относящегося к классу NSJSONSerialization. Объект, возвращаемый этим методом, будет представлять собой либо словарь, либо массив в зависимости от того, какие данные ему были переданы. Рассмотрим пример:

```
/* Сейчас попытаемся сериализовать объект JSON в словарь. */
error = nil;
id jsonObject = [NSJSONSerialization
                 JSONObjectWithData:jsonData
                 options:NSJSONReadingAllowFragments
                 error:&error];

if (jsonObject != nil &&
    error == nil){

  NSLog(@"Successfully deserialized...");

  if ([jsonObject isKindOfClass:[NSDictionary class]]){

    NSDictionary *deserializedDictionary = (NSDictionary *)jsonObject;
    NSLog(@"Deserialized JSON Dictionary = %@", deserializedDictionary);

  }
  else if ([jsonObject isKindOfClass:[NSArray class]]){

    NSArray *deserializedArray = (NSArray *)jsonObject;
    NSLog(@"Deserialized JSON Array = %@", deserializedArray);

  }

  else {
    /* Был возвращен какой-то другой объект. Мы не знаем,
       что делать в этой ситуации, так как десериализатор
       возвращает только словари или массивы. */
  }

}
else if (error != nil){
  NSLog(@"An error happened while deserializing the JSON data.");
}
```

Если теперь объединить этот код с кодом из раздела 8.9, то можно будет сначала сериализовать словарь в объект JSON, десериализовать объект JSON обратно в словарь, а потом вывести результаты на консоль, чтобы убедиться, что все работает нормально:

```
- (BOOL) application:(UIApplication *)application
didFinishLaunchingWithOptions:(NSDictionary *)launchOptions{
  NSDictionary *dictionary =
  @{
    @"First Name" : @"Anthony",
    @"Last Name" : @"Robbins",
    @"Age" : @51,
    @"Children" : @[
        @"Anthony's Son 1",
        @"Anthony's Daughter 1",
        @"Anthony's Son 2",
        @"Anthony's Son 3",
        @"Anthony's Daughter 2",
        ],
};

  NSError *error = nil;
  NSData *jsonData = [NSJSONSerialization
                      dataWithJSONObject:dictionary
                      options:NSJSONWritingPrettyPrinted
                      error:&error];

  if ([jsonData length] > 0 &&
      error == nil){

    NSLog(@"Successfully serialized the dictionary into data.");

    /* Сейчас попытаемся сериализовать объект JSON в словарь. */
    error = nil;
    id jsonObject = [NSJSONSerialization
                     JSONObjectWithData:jsonData
                     options:NSJSONReadingAllowFragments
                     error:&error];

    if (jsonObject != nil &&
        error == nil){

      NSLog(@"Successfully deserialized...");

      if ([jsonObject isKindOfClass:[NSDictionary class]]){

        NSDictionary *deserializedDictionary = (NSDictionary *)jsonObject;
        NSLog(@"Deserialized JSON Dictionary = %@", deserializedDictionary);

      }
      else if ([jsonObject isKindOfClass:[NSArray class]]){

        NSArray *deserializedArray = (NSArray *)jsonObject;
        NSLog(@"Deserialized JSON Array = %@", deserializedArray);

      }
```

```
        else {
        /* Был возвращен какой-то другой объект. Мы не знаем, что делать
           в этой ситуации, так как десериализатор возвращает только словари
           или массивы. */
        }

      }
      else if (error != nil){
        NSLog(@"An error happened while deserializing the JSON data.");
      }

    }
    else if ([jsonData length] == 0 &&
             error == nil){

      NSLog(@"No data was returned after serialization.");

    }
    else if (error != nil){

      NSLog(@"An error happened = %@", error);

    }
    self.window = [[UIWindow alloc] initWithFrame:
                   [[UIScreen mainScreen] bounds]];
    // Точка переопределения для дополнительной настройки после запуска приложения
    self.window.backgroundColor = [UIColor whiteColor];
    [self.window makeKeyAndVisible];
    return YES;
}
```

Параметр `options` метода `JSONObjectWithData:options:error:` принимает одно или несколько следующих значений:

- `NSJSONReadingMutableContainers` — словарь или массив, возвращенный методом `JSONObjectWithData:options:error:`, будет изменяемым. Иными словами, этот метод будет возвращать либо экземпляр `NSMutableArray`, либо экземпляр `NSMutableDictionary` в противоположность изменяемому массиву или словарю;
- `NSJSONReadingMutableLeaves` — листовые значения будут инкапсулированы в экземпляры `NSMutableString`;
- `NSJSONReadingAllowFragments` — обеспечивает десериализацию данных JSON, чей корневой объект верхнего уровня не является массивом или словарем.

## См. также

Раздел 11.9.

# 11.11. Включение в приложения функций социального обмена контентом

## Постановка задачи

Требуется предоставить в приложении функции социального обмена контентом. Например, у пользователя мобильного устройства должна быть возможность написать твит или обновить статус в Facebook.

## Решение

Внедрите в ваше приложение фреймворк `Social` и воспользуйтесь классом `SLComposeViewController` для обеспечения социального обмена сообщениями, например твитами.

## Обсуждение

Класс `SLComposeViewController` входит в состав фреймворка Social. Он приспособлен к работе с модулями компилятора LLVM. Чтобы приступить к использованию этого фреймворка, вам всего лишь потребуется импортировать в проект обобщающий заголовок, вот так:

```
#import "ViewController.h"
#import <Social/Social.h>

@implementation ViewController
```

Поскольку Apple обогащает свой SDK новыми возможностями социального обмена контентом, вы можете запрашивать фреймворк `Social` и прямо во время выполнения узнавать, какой из сервисов доступен на устройстве, на котором работает ваше приложение. Поскольку набор таких сервисов может варьироваться от устройства к устройству, перед попыткой использовать тот или иной сервис обязательно следует убедиться, что нужный сервис работает. Чтобы запросить у iOS такую информацию, воспользуйтесь методом класса `isAvailableForServiceType:`, относящимся к классу `SLComposeViewController`. Параметр, передаваемый этому методу, относится к типу `NSString`, а вот список некоторых валидных параметров, которые можно передать этому методу:

- `SOCIAL_EXTERN NSString *const SLServiceTypeTwitter;`
- `SOCIAL_EXTERN NSString *const SLServiceTypeFacebook;`
- `SOCIAL_EXTERN NSString *const SLServiceTypeSinaWeibo;`
- `SOCIAL_EXTERN NSString *const SLServiceTypeTencentWeibo;`
- `SOCIAL_EXTERN NSString *const SLServiceTypeLinkedIn.`

Убедившись, что нужный сервис доступен, вы можете воспользоваться методом класса `composeViewControllerForServiceType:`, относящимся к классу `SLComposeViewController`. Так вы получаете новый экземпляр контроллера вида для социального обмена. Далее все совсем просто. Вам потребуется всего лишь использовать в контроллере для социального обмена один или несколько следующих методов:

- `setInitialText:` — задает строку, которой вы хотите поделиться;
- `addImage:` — добавляет изображение, которое должно прикрепляться к вашему посту;
- `addURL:` — добавляет URL, которым можно делиться наряду с текстом и изображением.

Экземпляр класса `SLComposeViewController` также обладает очень удобным свойством `completionHandler`. Оно представляет собой блоковый объект типа `SLComposeViewControllerCompletionHandler`. Этот обработчик завершения будет вызываться всякий раз, когда пользователь успешно завершает процесс обмена контентом (то есть пользователь успешно отправляет пост, который iOS доставляет на сайт Twitter, Facebook и др.) либо закрывает диалоговое окно. Этому методу будет передаваться параметр типа `SLComposeViewControllerResult`. Он обозначает тип произошедшего события — например, успех или отмену операции.

Итак, довольно слов, переходим к сути. Далее будет рассмотрен фрагмент кода, который пытается определить, обладает ли данное устройство возможностями социального обмена контентом через Twitter. Если это так, код создает простой твит с картинкой и URL, после чего отображает для пользователя диалоговое окно Twitter, готовое к отправке сообщения:

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

  if ([SLComposeViewController
       isAvailableForServiceType:SLServiceTypeTwitter]){

    SLComposeViewController *controller =
    [SLComposeViewController
    composeViewControllerForServiceType:SLServiceTypeTwitter];

    [controller setInitialText:@"MacBook Airs are amazingly thin!"];
    [controller addImage:[UIImage imageNamed:@"MacBookAir"]];
    [controller addURL:[NSURL URLWithString:@"http://www.apple.com/"]];

    controller.completionHandler = ^(SLComposeViewControllerResult result){
      NSLog(@"Completed");
    };

    [self presentViewController:controller animated:YES completion:nil];

  } else {
    NSLog(@"The twitter service is not available");
  }
}
```

Запустив это приложение на устройстве, где поддерживается работа с Twitter (такая интеграция обеспечивается с помощью соответствующих настроек iOS), вы увидите картинку, напоминающую рис. 11.2.

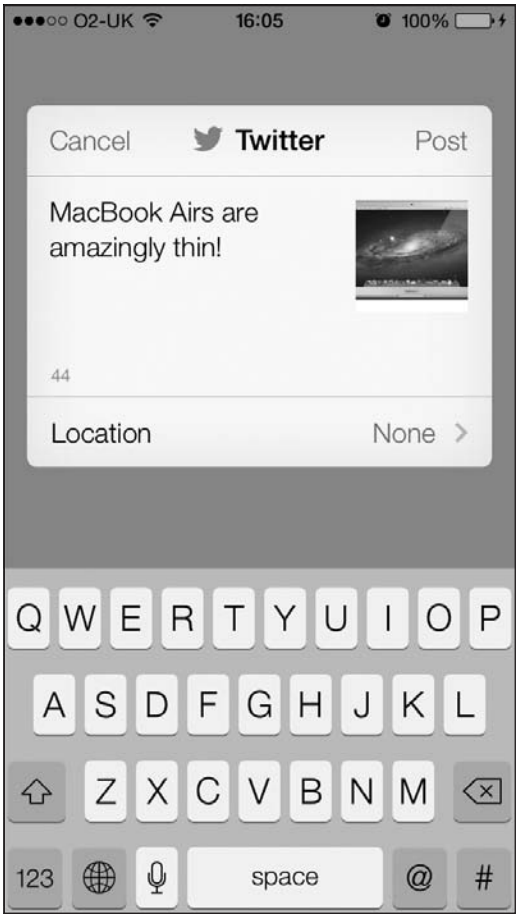


**Рис. 11.2.** Создание простого твита с помощью фреймворка Social

Обладая этой информацией, мы можем создавать и разные другие сообщения — например, обновления для учетной записи Facebook. На самом деле, как объяснялось ранее, вам всего лишь потребуется определить во время исполнения, активизирован ли на устройстве искомый сервис, а потом попытаться воспользоваться им — добавить в запросе текст, изображения, URL.

Наконец, не забывайте, что обработчики завершения для ваших видов, используемых для составления таких сообщений, *могут* вызываться не в том потоке, в котором вы создавали контроллер. Итак, помня об этом, пользуйтесь приемами, изученными в главе 7, и переключайтесь на работу с главным потоком внутри обработчика завершения, если собираетесь выполнять что-либо, имеющее отношение к пользовательскому интерфейсу.

## См. также

Раздел 11.0.

# 11.12. Синтаксический разбор XML с помощью NSXMLParser

## Постановка задачи

Необходимо выполнить синтаксический разбор (парсинг) фрагмента кода на языке XML или XML-документа.

## Решение

Воспользуйтесь классом `NSXMLParser`.

## Обсуждение

Для синтаксического разбора XML-содержимого класс `NSXMLParser` использует делегат. Создадим простой XML-файл, содержащий следующие данные (сохраните этот файл в вашем проекте как `MyXML.xml`):

```
<?xml version="1.0" encoding="UTF-8"?>
<root>

  <person id="1">
    <firstName>Anthony</firstName>
    <lastName>Robbins</lastName>
    <age>51</age>
  </person>

  <person id="2">
    <firstName>Richard</firstName>
    <lastName>Branson</lastName>
    <age>61</age>
  </person>

</root>
```

Теперь определим свойство типа `NSXMLParser`:

```
#import "AppDelegate.h"

@interface AppDelegate () <NSXMLParserDelegate>
@property (nonatomic, strong) NSXMLParser *xmlParser;
@end

@implementation AppDelegate
```

Кроме того, как видите, я определил делегат моего приложения как делегат XML-парсера, который подчиняется протоколу NSXMLParserDelegate. Согласно этому протоколу, объект делегата XML-парсера должен относиться к типу NSXMLParser. Считаем с диска файл MyXML.xml и передадим его на обработку в XML-парсер:

```
- (BOOL)            application:(UIApplication *)application
  didFinishLaunchingWithOptions:(NSDictionary *)launchOptions{

  NSString *xmlFilePath = [[NSBundle mainBundle] pathForResource:@"MyXML"
                                                          ofType:@"xml"];

  NSData *xml = [[NSData alloc] initWithContentsOfFile:xmlFilePath];

  self.xmlParser = [[NSXMLParser alloc] initWithData:xml];
  self.xmlParser.delegate = self;
  if ([self.xmlParser parse]){
    NSLog(@"The XML is parsed.");
  } else{
    NSLog(@"Failed to parse the XML");
  }

  self.window = [[UIWindow alloc] initWithFrame:
                 [[UIScreen mainScreen] bounds]];

  self.window.backgroundColor = [UIColor whiteColor];
  [self.window makeKeyAndVisible];
  return YES;
}
```

Сначала считываем содержимое файла в экземпляр NSData, а потом инициализируем XML-парсер с помощью метода initWithData:, используя данные, считанные из XML-файла. Затем вызываем метод parse XML-парсера, чтобы запустить процесс синтаксического разбора. Этот метод заблокирует актуальный поток до тех пор, пока синтаксический разбор не завершится. Если вам требуется произвести синтаксический разбор больших XML-файлов, используйте для этого глобальную диспетчерскую очередь.

Для синтаксического разбора XML-файла необходимо знать методы делегатов, определенные в протоколе NSXMLParserDelegate, а также понимать, за что они отвечают:

- parserDidStartDocument: — вызывается при запуске синтаксического разбора;
- parserDidEndDocument: — вызывается по окончании синтаксического разбора;
- parser:didStartElement:namespaceURI:qualifiedName:attributes: — вызывается, когда парсер встречает и начинает разбирать новый элемент в XML-документе;
- parser:didEndElement:namespaceURI:qualifiedName: — вызывается, когда парсер завершает синтаксический разбор текущего элемента;
- parser:foundCharacters: — вызывается, когда парсер анализирует строковое содержимое элементов.

С помощью этих методов делегата можно определить объектную модель для XML-объектов. Сначала определим объект, который будет представлять XML-элемент. Сделаем это в классе `XMLElement`:

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

@interface XMLElement : NSObject

@property (nonatomic, strong) NSString *name;
@property (nonatomic, strong) NSString *text;
@property (nonatomic, strong) NSDictionary *attributes;
@property (nonatomic, strong) NSMutableArray *subElements;
@property (nonatomic, weak) XMLElement *parent;

@end
```

Теперь реализуем класс `XMLElement`:

```
#import "XMLElement.h"

@implementation XMLElement

- (NSMutableArray *) subElements{
  if (subElements == nil){
    subElements = [[NSMutableArray alloc] init];
  }
  return subElements;
}

@end
```

Мы хотим, чтобы изменяемый массив `subElements` создавался лишь тогда, когда при достижении этой точки в коде мы имеем значение `nil`. Поэтому код для выделения и инициализации свойства `subElements` класса `XMLElement` поместим в его собственном методе-получателе. Если у XML-элемента нет дочерних элементов, то использовать это свойство не придется. Ведь отсутствует точка, в которой можно было бы выделить и инициализировать изменяемый массив для данного элемента. Такая техника называется «ленивое выделение» (Lazy Allocation).

Итак, продолжим. Определим экземпляр `XMLElement` и назовем его `rootElement`. Наш план — начать синтаксический разбор и подробно изучить XML-файл по мере разбора его и методов его делегата, пока не рассмотрим весь файл целиком:

```
#import "AppDelegate.h"
#import "XMLElement.h"
@interface AppDelegate () <NSXMLParserDelegate>

@property (nonatomic, strong) UIWindow *window;
@property (nonatomic, strong) NSXMLParser *xmlParser;
```

```
@property (nonatomic, strong) XMLElement *rootElement;
@property (nonatomic, strong) XMLElement *currentElementPointer;

@end
@implementation AppDelegate
```

currentElementPointer будет соответствовать тому XML-элементу, который мы в данный момент разбираем в XML-структуре. В ходе синтаксического разбора можно будет перемещаться по этой структуре вверх и вниз. В отличие от постоянно изменяющегося указателя currentElementPointer, rootElement всегда будет оставаться корневым элементом XML-файла и его значение не изменится в ходе синтаксического разбора данного файла.

Начнем синтаксический разбор. Первый элемент, который нас интересует, — это метод parserDidStartDocument:. В нем мы просто сбрасываем все значения:

```
- (void)parserDidStartDocument:(NSXMLParser *)parser{
  self.rootElement = nil;
  self.currentElementPointer = nil;
}
```

Следующий метод называется parser:didStartElement:namespaceURI:qualifiedName:attributes:. В этом методе создадим корневой элемент (если он еще не создан). Когда в XML-файле начинается разбор любого нового элемента, мы вычисляем, где именно в структуре XML-файла находимся, а потом добавляем новый элемент-объект к актуальному элементу-объекту:

```
- (void)   parser:(NSXMLParser *)parser
  didStartElement:(NSString *)elementName
     namespaceURI:(NSString *)namespaceURI
     qualifiedName:(NSString *)qName
       attributes:(NSDictionary *)attributeDict{

  if (self.rootElement == nil){
    /* У нас нет корневого элемента. Создадим такой элемент
       и укажем на него. */
    self.rootElement = [[XMLElement alloc] init];
    self.currentElementPointer = self.rootElement;
  } else {
    /* Корневой элемент уже есть. Создаем новый элемент и добавляем его
       в качестве одного из дочерних элементов текущего элемента. */
    XMLElement *newElement = [[XMLElement alloc] init];
    newElement.parent = self.currentElementPointer;
    [self.currentElementPointer.subElements addObject:newElement];
    self.currentElementPointer = newElement;
  }

  self.currentElementPointer.name = elementName;
  self.currentElementPointer.attributes = attributeDict;

}
```

Теперь перед нами метод `parser:foundCharacters:`. Для каждого текущего элемента этот метод может вызываться несколько раз, поэтому необходимо гарантировать, что мы сможем сделать несколько записей в этом методе. Например, если текст элемента имеет 4000 символов в длину, то парсер может разобрать не более 1000 символов за первый ход, еще 1000 — за второй и т. д. В таком случае синтаксический анализатор вызовет метод `parser:foundCharacters:` для данного элемента четыре раза. Вероятно, вам потребуется просто аккумулировать результаты и вернуть их в виде строки:

```
- (void)        parser:(NSXMLParser *)parser
       foundCharacters:(NSString *)string{

  if ([self.currentElementPointer.text length] > 0){
    self.currentElementPointer.text =
      [self.currentElementPointer.text stringByAppendingString:string];
  } else {
    self.currentElementPointer.text = string;
  }

}
```

Следующий метод, с которым необходимо разобраться, называется `parser:didEndElement:namespaceURI:qualifiedName:`. Он вызывается, когда парсер доходит до конца элемента. Здесь нам нужно просто вернуть указатель XML-элементов на уровень выше, к тому элементу, который является родительским для только что проанализированного. Все довольно просто:

```
- (void) parser:(NSXMLParser *)parser
  didEndElement:(NSString *)elementName
   namespaceURI:(NSString *)namespaceURI
  qualifiedName:(NSString *)qName{

  self.currentElementPointer = self.currentElementPointer.parent;

}
```

И последний, но немаловажный момент. Нужно также обработать метод `parserDidEndDocument:` и избавиться от текущего свойства `currentElementPointer`:

```
- (void)parserDidEndDocument:(NSXMLParser *)parser{
  self.currentElementPointer = nil;
}
```

Вот и все. Теперь можете выполнить синтаксический разбор нашего документа:

```
- (BOOL)            application:(UIApplication *)application
  didFinishLaunchingWithOptions:(NSDictionary *)launchOptions{

  NSString *xmlFilePath = [[NSBundle mainBundle] pathForResource:@"MyXML"
                                                          ofType:@"xml"];
```

```
  NSData *xml = [[NSData alloc] initWithContentsOfFile:xmlFilePath];

  self.xmlParser = [[NSXMLParser alloc] initWithData:xml];
  self.xmlParser.delegate = self;
  if ([self.xmlParser parse]){
    NSLog(@"The XML is parsed.");

    /* self.rootElement сейчас является корневым элементом XML-документа. */

  } else{
    NSLog(@"Failed to parse the XML");
  }

  self.window = [[UIWindow alloc] initWithFrame:
                 [[UIScreen mainScreen] bounds]];

  self.window.backgroundColor = [UIColor whiteColor];
  [self.window makeKeyAndVisible];
  return YES;
}
```

Теперь можно использовать свойство `rootElement` для обхода всей структуры нашего XML-документа.

# 12 Управление файлами и каталогами

## 12.0. Введение

Операционная система iOS основана на MacOS X, которая, в свою очередь, построена на базе операционной системы Unix. В iOS полная структура каталогов остается невидимой для приложения, поскольку каждое приложение, написанное iOS-разработчиком, существует в собственной песочнице. Эта защищенная среда не случайно называется песочницей: она действительно представляет собой жестко ограниченную область, и содержимое каталога песочницы доступно лишь тому приложению, которое владеет ею. У каждого приложения есть собственный каталог-песочница, по умолчанию у таких песочниц есть подкаталоги, к которым могут обращаться приложения.

Когда приложение для iOS устанавливается на устройстве, система создает для этого приложения структуру каталогов (рис. 12.1).

- Name.app — несмотря на странное название с расширением .app, это каталог. Все содержимое вашего основного пакета оказывается здесь. Например, все пиктограммы приложения, двоичный файл приложения, различные брендинговые изображения, шрифты, звуки и пр. автоматически отправятся в этот каталог, когда система iOS будет устанавливать приложение на устройстве. Name — это имя продукта, которое вы задали для приложения. Итак, если вы назвали приложение `MyApp`, то его каталог `.app` будет называться `MyApp.app`.
- Documents/ — этот каталог является местом назначения для всего контента, создаваемого пользователем. Содержимое, которое заполняется, скачивается или создается вашим приложением, не должно храниться в этом каталоге.
- Library/ — этот каталог используется для хранения кэшированных файлов, пользовательских настроек и т. д. Как правило, он не содержит никаких файлов, а только подкаталоги, в которых уже находятся другие файлы.

В корневом каталоге каждого приложения содержатся различные другие каталоги, о которых я расскажу далее.

- Library/Caches/ — в этом каталоге хранится информация, которую ваше приложение позже сможет воссоздать, если возникнет такая необходимость. iOS

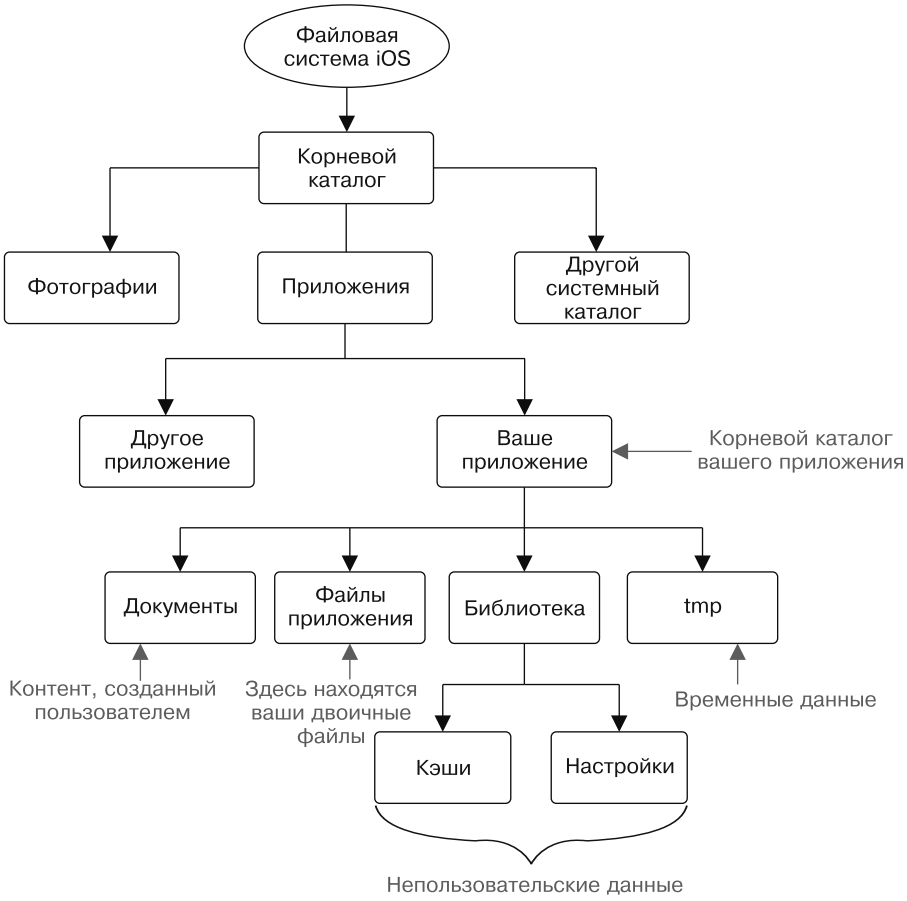


**Рис. 12.1.** Структура файловой системы в iOS

не выполняет резервного копирования этого каталога. Кроме того, iOS может удалить его содержимое, если дисковое пространство заканчивается, а ваше приложение в данный момент не работает! Поэтому работа приложения не должна слишком зависеть от этого каталога — будьте готовы к тому, что его содержимое потребуется создавать заново. Повторю: iOS не выполняет резервного копирования информации из этого каталога, так что, когда работа вашего приложения приостановлена, данная информация вполне может быть удалена.

Если работа приложения напрямую зависит от файлов и каталогов, которые должны создаваться на диске, то этот каталог не лучшее место для хранения информации. Такую важную информацию лучше хранить в папке `/tmp`.

- Library/Preferences/ — как понятно из названия, в этом приложении хранятся настройки, которые приложение должно запоминать и активизировать от запуска к запуску. Мы подробнее поговорим об этом в дальнейшем. iOS выполняет резервное копирование информации из этого каталога.

- Library/Application Support/ — данные, создаваемые вашим приложением (за исключением тех данных, которые создает сам пользователь), должны храниться в этом каталоге. Приятно отметить, что iOS выполняет резервное копирование данного каталога. Возможно, этот каталог не будет создаваться автоматически и вам придется создавать его самостоятельно, если его не существует. О создании этого каталога мы поговорим позже в данной главе.
- tmp/ — это временные файлы, которые ваше приложение может скачивать, создавать и т. д. iOS не выполняет резервного копирования данного каталога. Например, вы можете скачать из Интернета несколько фотографий и сохранить их в каталоге, чтобы повысить производительность вашего приложения, ведь в таком случае эти фотографии не придется скачивать заново при каждом запуске приложения. Каталог служит именно для этой цели. Убедитесь, что в нем не будут храниться документы или файлы, создаваемые пользователем.

Итак, теперь вы знаете, какие каталоги автоматически создаются операционной системой при установке приложения на устройстве с iOS. Далее найдем пути к остальным полезным каталогам, которые мы здесь уже упоминали. Воспользуемся теми API, которые Apple предоставляет специально для этих целей (о таких API пойдет речь в дальнейшем в этой главе).

# 12.1. Определение пути к самым полезным каталогам на диске

## Постановка задачи

Требуется определить путь к некоторым наиболее полезным каталогам, доступ к которым есть у вашего приложения (например, к каталогам, рассмотренным во введении). Мы должны знать эти пути, чтобы иметь возможность обращаться к каталогам или создавать в них новое содержимое.

Программистам требуется использовать API, предоставляемые в iOS SDK, для нахождения путей к каталогам и/или файлам. Иными словами, путь к файлу или каталогу никогда не следует угадывать. Если, например, вы ищете пути, один из которых ведет к каталогу Documents (Документы), то нужно гарантировать, что для этого применяются правильные API. Никогда, ни в коем случае не рассчитывайте на то, что этот каталог будет называться в пакете вашего приложения именно Documents и никак иначе. Для нахождения пути к этому каталогу достаточно воспользоваться подходящими API. Если вас интересует не сам каталог, а отдельные содержащиеся в нем файлы, добавьте к концу обнаруженного пути имена тех или иных файлов.

## Решение

Используйте метод экземпляра `URLsForDirectory:inDomains:`, относящийся к классу `NSFileManager`.

## Обсуждение

Класс NSFileManager обеспечивает множество операций, связанных с файлами и каталогами, выполняемых в iOS. Все такие операции осуществляются прямо внутри ваших приложений, от вас требуется всего лишь создать экземпляр этого класса. Не рекомендую пользоваться разделяемым файловым менеджером, который предоставляется этим классом с помощью метода класса defaultManager, поскольку этот менеджер не является потокобезопасным. Лучше самостоятельно создать экземпляр класса NSFileManager и управлять им.

Метод экземпляра URLsForDirectory:inDomains:, относящийся к классу NSFileManager, позволяет искать конкретные каталоги в файловой системе iOS, в основном в песочнице вашего приложения. Этот метод имеет два параметра:

- URLsForDirectory: — это каталог, который вы хотите найти. Передайте этому параметру значение типа NSSearchPathDirectory (оно является перечислением). Далее поговорим о нем подробнее;
- inDomains — параметр указывает, где вы собираетесь искать конкретный каталог. Значение этого параметра должно относиться к типу NSSearchPathDomainMask (это тоже перечисление).

Предположим, вы хотите найти путь к каталогу Documents (Документы) вашего приложения. Вот как просто это делается:

```
NSFileManager *fileManager = [[NSFileManager alloc] init];
NSArray *urls = [fileManager URLsForDirectory:NSDocumentDirectory
                                    inDomains:NSUserDomainMask];

if ([urls count] > 0){
  NSURL *documentsFolder = urls[0];
  NSLog(@"%@", documentsFolder);
} else {
  NSLog(@"Could not find the Documents folder.");
}
```

Как видите, создав собственный экземпляр NSFileManager, мы передали значение NSDocumentDirectory в качестве искомого каталога и NSUserDomainMask — в качестве области поиска. Рассмотрим некоторые наиболее важные значения, которые можно передать каждому из параметров метода экземпляра URLsForDirectory:inDomains:, относящегося к классу NSFileManager:

- URLsForDirectory;
- NSLibraryDirectory — библиотечный каталог приложения;
- NSCachesDirectory — каталог кэша — о нем рассказано ранее;
- NSDocumentDirectory — каталог документов;
- inDomains;
- NSUserDomainMask.

Данное значение указывает, что поиск должен выполняться в актуальном пользовательском каталоге. В системе OS X этот каталог обозначался бы ~/.

С помощью этого метода мы можем найти и другие каталоги, например caches, как показано далее:

```
NSFileManager *fileManager = [[NSFileManager alloc] init];
NSArray *urls = [fileManager URLsForDirectory:NSCachesDirectory
                                    inDomains:NSUserDomainMask];

if ([urls count] > 0){
  NSURL *cachesFolder = urls[0];
  NSLog(@"%@", cachesFolder);
} else {
  NSLog(@"Could not find the Caches folder.");
}
```

Если вы хотите найти каталог tmp, воспользуйтесь функцией NSTemporaryDirectory() на языке C, вот так:

```
NSString *tempDirectory = NSTemporaryDirectory();
NSLog(@"Temp Directory = %@", tempDirectory);
```

Выполнив эту команду на устройстве, получим примерно следующий вывод:

```
Temp Directory = /private/var/mobile/
Applications/<# Здесь находится ID вашего приложения #>/tmp/
```

## См. также

Раздел 12.0.

# 12.2. Запись информации в файлы и считывание информации из файлов

## Постановка задачи

Требуется сохранить на диске информацию (например, текст, данные, изображения и т. д.).

## Решение

Все классы Cocoa, обеспечивающие сохранение информации, например NSString, UIImage и NSData, предоставляют методы экземпляра, позволяющие сохранять данные на диске по заданному пути.

## Обсуждение

Чтобы сохранять текст на диске (предполагается, что ваш текст сохранен в экземпляре NSString или неизменяемой версии этого класса), можно воспользоваться методом экземпляра writeToFile:atomically:encoding:error:, относящимся к этому

классу. Этот метод применяется со строками, представляющими собой пути назначения. Вот его отдельные параметры.

- writeToFile — путь к файлу, в который нужно записать информацию, указывается в виде строки.
- atomically — логическое значение. Если оно установлено в YES, то файл сначала будет записываться во временное пространство, а потом перемещаться на тот адрес, где вы хотите его расположить. Так гарантируется, что содержимое файла, которое требуется сохранить, сначала будет просто перенесено на диск, а уже затем пересохранено в месте назначения. Поэтому, если вдруг отказ системы iOS произойдет прежде, чем файл будет сохранен в месте назначения, контент будет доступен и позднее, когда операционная система возобновит работу. И вы сможете сохранить информацию куда следует. При сохранении информации рекомендуется устанавливать данное значение в YES, чтобы ни при каких обстоятельствах не терять информацию работающего приложения безвозвратно.
- encoding — кодировка текста, который вы хотите сохранить по указанному адресу. Обычно в данном случае используется кодировка UTF-8, задаваемая с помощью константы NSUTF8StringEncoding.
- error — принимает указатель на объект NSError. Поэтому если операция сохранения завершится ошибкой и будет прервана, то вы сможете выяснить, какая именно ошибка произошла. Этому параметру можно передать значение nil, если вас не интересуют ошибки, которые могут возникнуть в процессе сохранения. Не забывайте, что эта функция возвращает логическое значение и вы можете воспользоваться им, чтобы просто определить, произошла какая-либо ошибка или нет.

Например, если у вас есть некий текст, который вы хотите сохранить в приложении, но резервно копировать его на уровне системы iOS не требуется, то можно поступить так:

```
NSString *someText = @"Random string that won't be backed up.";

NSString *destinationPath =
[NSTemporaryDirectory()
  stringByAppendingPathComponent:@"MyFile.txt"];

NSError *error = nil;
BOOL succeeded = [someText writeToFile:destinationPath
                            atomically:YES
                              encoding:NSUTF8StringEncoding
                                 error:&error];

if (succeeded) {
  NSLog(@"Successfully stored the file at: %@", destinationPath);
} else {
  NSLog(@"Failed to store the file. Error = %@", error);
}
```

Кроме того, когда сделаете все это, можете дополнительно убедиться, что вся работа выполнена верно. Попытайтесь считать ту же строку из файла назначения в память. Для этого используется метод класса `stringWithContentsOfFile:encoding:error:`, относящийся к классу `NSString`. В ответ вы должны получить автоматически высвобожденную строку, которая представляет собой содержимое указанного файла. Если вы хотите явно инстанцировать объект типа `NSString` с содержимым файла, просто примените метод экземпляра `initWithContentsOfFile:encoding:error:`, относящийся к классу `NSString`, вот так:

```
- (BOOL) writeText:(NSString *)paramText toPath:(NSString *)paramPath{
  return [paramText writeToFile:paramPath
                     atomically:YES
                       encoding:NSUTF8StringEncoding
                          error:nil];
}

- (NSString *) readTextFromPath:(NSString *)paramPath{
  return [[NSString alloc] initWithContentsOfFile:paramPath
                                         encoding:NSUTF8StringEncoding
                                            error:nil];

}

- (BOOL) application:(UIApplication *)application
didFinishLaunchingWithOptions:(NSDictionary *)launchOptions{

  NSString *filePath = [NSTemporaryDirectory()
  stringByAppendingPathComponent:@"MyFile.txt"];

  if ([self writeText:@"Hello, World!" toPath:filePath]){

    NSString *readText = [self readTextFromPath:filePath];
    if ([readText length] > 0){
      NSLog(@"Text read from disk = %@", readText);
    } else {
      NSLog(@"Failed to read the text from disk.");
    }

  } else {
    NSLog(@"Failed to write the file.");
  }

  self.window = [[UIWindow alloc]
                 initWithFrame:[[UIScreen mainScreen] bounds]];
  self.window.backgroundColor = [UIColor whiteColor];
  [self.window makeKeyAndVisible];
  return YES;
}
```

Здесь мы создали два удобных метода, позволяющих нам записывать текст и считывать его из указанного места. Затем используем эти методы в делегате

нашего приложения, чтобы записать определенный текст в каталог temp, а потом считаем этот текст обратно в память и так убедимся, что методы работают нормально.

Если вы хотите работать с URL, инкапсулированными в экземпляры NSURL (или в экземпляры изменяемой версии этого класса), используйте в данном случае метод экземпляра writeToURL:atomically:encoding:error:.

Экземпляры NSURL могут указывать на ресурсы (файлы, каталоги и т. д.), расположенные в локальной системе или на удаленных устройствах. Так, экземпляр NSURL может представлять локальный файл в каталоге Documents (Документы) в вашем приложении, а другой NSURL — соответствовать URL сайта www.apple.com. Этот класс просто предоставляет вам функции, необходимые для доступа к URL и для работы с ними, независимо от типа конкретного URL.

Другие основополагающие классы обладают примерно такими же методами, как и NSString. Возьмем, к примеру, NSArray. Чтобы сохранить содержимое массива, пользуйтесь методом экземпляра writeToFile:atomically:, относящимся к классу NSArray. Чтобы считать с диска содержимое любого массива, можно просто выделить экземпляр массива, а потом инициализировать его с помощью initWithContentsOfFile: — это метод-инициализатор для работы с массивами. Вот примеры использования обоих методов:

```
NSString *filePath = [NSTemporaryDirectory()
                      stringByAppendingPathComponent:@"MyFile.txt"];

NSArray *arrayOfNames = @[@"Steve", @"John", @"Edward"];
if ([arrayOfNames writeToFile:filePath atomically:YES]){

  NSArray *readArray = [[NSArray alloc] initWithContentsOfFile:filePath];
  if ([readArray count] == [arrayOfNames count]){
    NSLog(@"Read the array back from disk just fine.");
  } else {
    NSLog(@"Failed to read the array back from disk.");
  }
} else {
  NSLog(@"Failed to save the array to disk.");
}
```

Метод экземпляра writeToFile:atomically:, относящийся к классу NSArray, может сохранять лишь массивы, содержащие объекты следующих типов:

- NSString;
- NSDictionary;
- NSArray;
- NSData;
- NSNumber;
- NSDate.

Если вы попытаетесь вставить в массив какие-либо другие объекты, то ваши данные не будут сохранены на диске, поскольку этот метод первым делом проверяет, относятся ли все объекты в составе массива к одному из вышеупомянутых типов. Это делается по той простой причине, что в противном случае среда времени исполнения Objective-C просто не разберется, как сохранять данные на диске. Предположим, мы инстанцируем класс под названием Person, создаем для этого класса два свойства, одно из которых соответствует имени, другое — фамилии. Затем инстанцируем экземпляр этого класса и добавим его к массиву. Как же массив сможет сохранить эту информацию о персоне на диск? Никак, поскольку система не будет знать, что именно требуется сохранять. Эта проблема называется «маршалинг». В операционной системе iOS она решена только для перечисленных типов.

Словари также очень похожи на массивы. Сохранение их данных на диске и считывание информации из словаря происходит практически так же, как и в случае с массивами. Имена методов такие же, как и в предыдущем примере, правила сохранения словарей не отличаются от правил сохранения массивов. Вот пример:

```
NSString *filePath = [NSTemporaryDirectory()
                      stringByAppendingPathComponent:@"MyFile.txt"];

NSDictionary *dict = @{
                       @"first name" : @"Steven",
                       @"middle name" : @"Paul",
                       @"last name" : @"Jobs",
                       };

if ([dict writeToFile:filePath atomically:YES]){
  NSDictionary *readDictionary = [[NSDictionary alloc]
                                  initWithContentsOfFile:filePath];

  /* Теперь сравним словари и проверим, является ли словарь, считываемый
   нами с диска, тем самым, который мы сохранили на диске */
  if ([readDictionary isEqualToDictionary:dict]){
    NSLog(@"The file we read is the same one as the one we saved.");
  } else {
    NSLog(@"Failed to read the dictionary from disk.");
  }
} else {
  NSLog(@"Failed to write the dictionary to disk.");
}
```

Как видите, в этом примере словарь записывается на диск, после чего считывается из этого самого места. После считывания мы сравниваем реальный словарь с тем, который сохранили на диске. Так мы должны убедиться, что оба словаря содержат одни и те же данные.

До сих пор мы применяли для сохранения содержимого на диске высокоуровневые классы, например `NSString` и `NSArray`. А что, если потребуется сохранить необработанный массив байтов? Это тоже делается просто. Предположим, у нас есть массив из четырех символов и его требуется сохранить на диск:

```
char bytes[4] = {'a', 'b', 'c', 'd'};
```

Чтобы сохранить этот необработанный массив байтов на диске простейшим способом, достаточно инкапсулировать его в другой высокоуровневой структуре данных, например NSData, а потом пользоваться соответствующими методами NSData для считывания данных с диска и записи их на диск. Методы сохранения и загрузки данных, применяемые в классе NSData, практически идентичны соответствующим методам классов NSArray и NSDictionary. Вот пример сохранения необработанных данных на диске и считывания их с диска:

```
NSString *filePath = [NSTemporaryDirectory()
                      stringByAppendingPathComponent:@"MyFile.txt"];

char bytes[4] = {'a', 'b', 'c', 'd'};

NSData *dataFromBytes = [[NSData alloc] initWithBytes:bytes
                                               length:sizeof(bytes)];

if ([dataFromBytes writeToFile:filePath atomically:YES]){
  NSData *readData = [[NSData alloc] initWithContentsOfFile:filePath];
  if ([readData isEqualToData:dataFromBytes]){
    NSLog(@"The data read is the same data as was written to disk.");
  } else {
    NSLog(@"Failed to read the data from disk.");
  }
} else {
  NSLog(@"Failed to save the data to disk.");
}
```

## См. также

Раздел 12.0.

# 12.3. Создание каталогов на диске

## Постановка задачи

Требуется возможность создавать на диске каталоги и сохранять в них определенные файлы из вашего приложения.

## Решение

Пользуйтесь методом экземпляра createDirectoryAtPath:withIntermediateDirectories:attributes:error:, относящимся к классу NSFileManager, как показано далее:

```
- (BOOL) application:(UIApplication *)application
didFinishLaunchingWithOptions:(NSDictionary *)launchOptions{

  NSFileManager *fileManager = [[NSFileManager alloc] init];

  NSString *tempDir = NSTemporaryDirectory();
```

```
  NSString *imagesDir = [tempDir stringByAppendingPathComponent:@"images"];

  NSError *error = nil;
  if ([fileManager createDirectoryAtPath:imagesDir
           withIntermediateDirectories:YES
                            attributes:nil
                                 error:&error]){

    NSLog(@"Successfully created the directory.");

  } else {
    NSLog(@"Failed to create the directory. Error = %@", error);
  }

  self.window = [[UIWindow alloc]
                 initWithFrame:[[UIScreen mainScreen] bounds]];
  self.window.backgroundColor = [UIColor whiteColor];
  [self.window makeKeyAndVisible];
  return YES;
}
```

## Обсуждение

API, предоставляемые классом `NSFileManager`, очень просты в использовании. Ничего удивительного в том, что для сохранения каталогов на диске эти API требуют написания всего нескольких строк кода. На первый взгляд метод `createDirectoryAtPath:withIntermediateDirectories:attributes:error:` может показаться страшноватым, но на самом деле не все так плохо. В дальнейшем я расскажу о различных параметрах, которые можно передать этому методу:

- `createDirectoryAtPath` — путь к тому каталогу, который требуется создать;
- `withIntermediateDirectories` — логический параметр. Если он имеет значение `YES`, то перед созданием конечного каталога метод создаст и все промежуточные каталоги. Например, если вы хотите создать каталог `images`, вложенный в каталог `data`, который, в свою очередь, вложен в каталог `tmp`, а каталог `data` пока не создан, то можете приказать этому методу создать каталог `tmp/data/images/`, установив для параметра `withIntermediateDirectories` значение `YES`. В таком случае система создаст и каталог `data`, и каталог `images`;
- `attributes` — словарь атрибутов, который можно передать системе. Эти атрибуты будут определять детали создания каталога. Здесь мы не будем использовать этот параметр, чтобы не усложнять пример. Но тут вы можете изменять такую информацию, как дата и время внесения изменений, дата и время создания, а также при желании и другие атрибуты созданного каталога;
- `error` — данный параметр принимает указатель на объект-ошибку типа `NSObject`. Этот объект будет заполняться любыми ошибками, которые могут возникать в процессе создания каталога. В принципе, целесообразно передавать объект-ошибку в этом параметре, так что, если он завершится неудачно (возвратит `NO`), вы сможете обратиться к ошибке и определить, что именно пошло не так.

## См. также

Раздел 12.1.

# 12.4. Перечисление файлов и каталогов

## Постановка задачи

Вы хотите построить перечень подкаталогов, содержащихся в каталоге, либо построить список файлов, содержащихся в каталоге. Акт перечисления означает, что вы просто хотите найти все каталоги и/или файлы, расположенные внутри другого каталога.

## Решение

Используйте метод экземпляра `contentsOfDirectoryAtPath:error:`, относящийся к классу `NSFileManager`, как показано далее. В данном примере мы перечисляем все файлы, каталоги и символьные ссылки, расположенные в каталоге пакета с приложением:

```
- (BOOL)            application:(UIApplication *)application
  didFinishLaunchingWithOptions:(NSDictionary *)launchOptions{

  NSFileManager *fileManager = [[NSFileManager alloc] init];
  NSString *bundleDir = [[NSBundle mainBundle] bundlePath];

  NSError *error = nil;
  NSArray *bundleContents = [fileManager
                             contentsOfDirectoryAtPath:bundleDir
                             error:&error];

  if ([bundleContents count] > 0 &&
    error == nil){
    NSLog(@"Contents of the app bundle = %@", bundleContents);
  }
  else if ([bundleContents count] == 0 &&
    error == nil){
    NSLog(@"Call the police! The app bundle is empty.");
  }
  else {
    NSLog(@"An error happened = %@", error);
  }

  self.window = [[UIWindow alloc]
                 initWithFrame:[[UIScreen mainScreen] bounds]];
  // Точка переопределения для дополнительной настройки после запуска приложения
  self.window.backgroundColor = [UIColor whiteColor];
  [self.window makeKeyAndVisible];
  return YES;
}
```

## Обсуждение

В некоторых приложениях для iOS иногда требуется строить перечень содержимого каталога. Возможно, вы пока не вполне понимаете, зачем это может понадобиться, поэтому рассмотрим соответствующий пример. Допустим, пользователь хочет скачать из Интернета 10 изображений и кэшировать их в вашем приложении. Вы выполняете эту операцию и сохраняете их, допустим, в каталоге `tmp/images/`, который создали вручную. Затем пользователь закрывает ваше приложение и вновь открывает его, а вы хотите отобразить в пользовательском интерфейсе вашей программы список уже загруженных файлов-изображений (в табличном виде). Как это сделать? Ничего сложного. Вам всего лишь потребуется перечислить содержимое вышеупомянутого каталога с помощью класса `NSFileManager`. Как было показано в подразделе «Решение» данного раздела, метод экземпляра `contentsOfDirectoryAtPath:error:`, относящийся к классу `NSFileManager`, возвращает массив объектов `NSString`, которые и соответствуют файлам, подкаталогам и символьным ссылкам внутри заданного каталога. Тем не менее непросто определить, какой из этих объектов является файлом, какой — подкаталогом и т. д. Чтобы получить от файлового менеджера более детализированную информацию, вызовите метод `contentsOfDirec toryAtURL:includingPropertiesForKeys:options:error:`. Рассмотрим параметры, которые можно передавать этому методу.

- `contentsOfDirectoryAtURL` — путь к каталогу, который вы хотите просмотреть. Этот путь должен предоставляться как экземпляр `NSURL`. Не волнуйтесь, если не знаете, как построить этот экземпляр. Вскоре мы об этом поговорим.
- `includingPropertiesForKeys` — это массив свойств, которые система iOS должна выбирать для каждого файла, каталога или элемента, найденного в конкретной директории. Например, вы можете указать, что в результатах должна возвращаться дата создания каждого элемента. Эта информация должна возвращаться в составе приходящего к вам экземпляра URL (или в экземплярах NSURL, получаемых от фреймворка). Вот список некоторых наиболее важных значений, которые могут находиться в этом массиве:
  - `NSURLIsDirectoryKey` — позволяет постфактум определить, указывает ли один из возвращенных URL на каталог;
  - `NSURLIsReadableKey` — возвращает дату создания того элемента, который расположен по возвращенному URL;
  - `NSURLContentAccessDateKey` — в возвращаемых результатах передает дату последнего обращения к содержимому;
  - `NSURLContentModificationDateKey` — как понятно из названия, это значение позволяет определять дату последнего изменения информации, расположенной по возвращенному URL.
- `options` — для этого параметра можно передать только одно из двух значений: `0` или `NSDirectoryEnumerationSkipsHiddenFiles`. Если введено второе значение, то, как понятно из его названия, при построении перечня будут пропущены все скрытые элементы.

- `error` — ссылка на объект, в который будет записываться информация об ошибке, если методу не удастся выполнить стоящую перед ним задачу. Обычно целесообразно передавать этому методу объекты-ошибки, если есть такая возможность. Если какие-то ошибки и будут возникать, то такие объекты помогут вам более уверенно с ними справляться.

Теперь, когда мы значительно более полно контролируем перечисление элементов, построим перечень всех элементов из каталога .app и выведем даты создания, последнего изменения элемента и последнего обращения к нему. Кроме того, будем выводить информацию о том, является ли этот элемент скрытым, а также есть ли у нас право считывания конкретного файла. Наконец, мы также укажем, являются конкретные элементы каталогами или нет. Приступим:

```
- (NSArray *) contentsOfAppBundle{
  NSFileManager *manager = [[NSFileManager alloc] init];
  NSURL *bundleDir = [[NSBundle mainBundle] bundleURL];

  NSArray *propertiesToGet = @[
                               NSURLIsDirectoryKey,
                               NSURLIsReadableKey,
                               NSURLCreationDateKey,
                               NSURLContentAccessDateKey,
                               NSURLContentModificationDateKey
                               ];

  NSError *error = nil;
  NSArray *result = [manager contentsOfDirectoryAtURL:bundleDir
                     includingPropertiesForKeys:propertiesToGet
                     options:0
                     error:&error];

  if (error != nil){
    NSLog(@"An error happened = %@", error);
  }
  return result;
}

- (NSString *) stringValueOfBoolProperty:(NSString *)paramProperty
                                   ofURL:(NSURL *)paramURL{

  NSNumber *boolValue = nil;
  NSError *error = nil;
  [paramURL getResourceValue:&boolValue
                      forKey:paramProperty
                       error:&error];
  if (error != nil){
    NSLog(@"Failed to get property of URL. Error = %@", error);
  }
  return [boolValue isEqualToNumber:@YES] ? @"Yes" : @"No";
}
```

```
- (NSString *) isURLDirectory:(NSURL *)paramURL{
  return [self stringValueOfBoolProperty:NSURLIsDirectoryKey ofURL:paramURL];
}

- (NSString *) isURLReadable:(NSURL *)paramURL{
  return [self stringValueOfBoolProperty:NSURLIsReadableKey ofURL:paramURL];
}

- (NSDate *) dateOfType:(NSString *)paramType inURL:(NSURL *)paramURL{
  NSDate *result = nil;
  NSError *error = nil;
  [paramURL getResourceValue:&result
                      forKey:paramType
                       error:&error];
  if (error != nil){
    NSLog(@"Failed to get property of URL. Error = %@", error);
  }
  return result;
}

- (void) printURLPropertiesToConsole:(NSURL *)paramURL{

  NSLog(@"Item name = %@", [paramURL lastPathComponent]);

  NSLog(@"Is a Directory? %@", [self isURLDirectory:paramURL]);

  NSLog(@"Is Readable? %@", [self isURLReadable:paramURL]);

  NSLog(@"Creation Date = %@",
  [self dateOfType:NSURLCreationDateKey inURL:paramURL]);

  NSLog(@"Access Date = %@",
  [self dateOfType:NSURLContentAccessDateKey inURL:paramURL]);

  NSLog(@"Modification Date = %@",
  [self dateOfType:NSURLContentModificationDateKey inURL:paramURL]);

  NSLog(@"----------------------------------");
}

- (BOOL) application:(UIApplication *)application
didFinishLaunchingWithOptions:(NSDictionary *)launchOptions{

  NSArray *itemsInAppBundle = [self contentsOfAppBundle];
  for (NSURL *item in itemsInAppBundle){
    [self printURLPropertiesToConsole:item];
  }

  self.window = [[UIWindow alloc]
                 initWithFrame:[[UIScreen mainScreen] bounds]];
  // Точка переопределения для дополнительной настройки после запуска приложения
```

```
  self.window.backgroundColor = [UIColor whiteColor];
  [self.window makeKeyAndVisible];
  return YES;
}
```

Вывод этой программы получится примерно таким:

```
Item name = Assets.car
Is a Directory? No Is Readable? Yes
Creation Date = 2013-06-2516:12:53 +0000
Access Date = 2013-06-2516:12:53 +0000
Modification Date = 2013-06-2516:12:53 +0000
---------------------------------
Item name = en.lproj
Is a Directory? Yes
Is Readable? Yes
Creation Date = 2013-06-2516:12:53 +0000
Access Date = 2013-06-2516:15:02 +0000
Modification Date = 2013-06-2516:12:53 +0000
---------------------------------
Item name = Enumerating Files and Folders
Is a Directory? No Is Readable? Yes
Creation Date = 2013-06-2516:15:01 +0000
Access Date = 2013-06-2516:15:04 +0000
Modification Date = 2013-06-2516:15:01 +0000
---------------------------------
```

Говоря об этом приложении, необходимо отметить, что мы используем метод экземпляра getResourceValue:forKey:error:, относящийся к классу NSURL, чтобы получить значение каждого из ключей, запрашиваемых у файлового менеджера, — например, даты создания и последнего изменения элемента. Эти требования мы передаем файловому менеджеру и приказываем ему выбрать эту информацию. Затем, как только у нас будут нужные URL, воспользуемся вышеупомянутым методом для получения различных свойств от результирующих URL.

Итак, рассмотрим различные части приложения. Я просто объясню, что делает каждый из написанных нами методов.

- `contentsOfAppBundle` — этот метод выполняет поиск в каталоге .app и находит все его элементы (файлы, подкаталоги, символьные ссылки и др.), после чего возвращает результат в виде массива. Все элементы в этом массиве относятся к типу `NSURL` и содержат дату собственного создания, последнего изменения, а также другие атрибуты, рассмотренные ранее.
- `stringValueOfBoolProperty:ofURL:` — этот метод выбирает строковый эквивалент (`Yes` или `No`) логического свойства URL. Например, информация о том, указывает конкретный URL на каталог или нет, сохраняется как двоичное логическое значение. Однако если вы хотите вывести это логическое значение на консоль, то его нужно преобразовать в строку. Для каждого URL у нас есть два элемента запроса, которые будут возвращать экземпляры `NSNumber`. Каждый из этих экземпляров (`NSURLIsDirectoryKey` и `NSURLIsReadableKey`) содержит логическое значение.

Итак, нам не приходится писать этот код для преобразования дважды, поскольку есть специальные методы для преобразования NSNumber в строку Yes или No.

- isURLDirectory: — принимает URL и проверяет, является ли он каталогом. На внутрисистемном уровне этот метод использует метод stringValueOfBoolProperty:ofURL: и передает ему ключ NSURLIsDirectoryKey.
- isURLReadable: — определяет, обладает ли ваше приложение доступом на чтение по указанному URL. На внутрисистемном уровне этот метод также использует метод stringValueOfBoolProperty:ofURL: и передает ему ключ NSURLIsDirectoryKey.
- dateOfType:inURL: — поскольку мы собираемся просматривать у каждого URL, соответствующего NSDate, свойства трех типов, просто инкапсулируем в данный метод нужный для этого код. Метод будет принимать ключ и возвращать в URL дату, ассоциированную с конкретным ключом.

Ну вот и все. Вы научились перечислять каталоги и получать все элементы, расположенные в конкретном каталоге. Вы даже умеете получать различные атрибуты для разных элементов.

## См. также

Разделы 12.1 и 12.2.

# 12.5. Удаление файлов и каталогов

## Постановка задачи

Вы создали на диске ряд файлов и/или каталогов, и они вам больше не нужны. Вы хотите их удалить.

## Решение

Используйте один из двух методов экземпляра, removeItemAtPath:error: или removeItemAtURL:error:, относящихся к классу NSFileManager. Первый метод принимает путь как строку, а второй — как URL.

## Обсуждение

Пожалуй, удаление файлов и каталогов — одна из простейших операций, которые можно совершать в файловом менеджере. В iOS нужно обязательно помнить о том, где вы храните ваши файлы и каталоги, а когда хранить их больше не требуется — избавляться от файлов и каталогов. Например, создадим пять текстовых файлов в каталоге tmp/text, а когда закончим работу с ними — удалим эти файлы. Тем временем мы успеем перечислить содержимое каталога по состоянию до и после удаления. Будем делать перечень лишь для того, чтобы убедиться, что все работает

правильно. Как вы помните, на момент установки приложения каталог tmp/ существует, а каталог tmp/text — нет. Поэтому для начала потребуется создать второй каталог. Как только закончим работу с файлами, удалим и сам каталог:

```
/* Создаем каталог по заданному пути */
- (void) createFolder:(NSString *)paramPath{
  NSError *error = nil;
  if ([self.fileManager createDirectoryAtPath:paramPath
                  withIntermediateDirectories:YES
                                   attributes:nil
                                        error:&error] == NO){
    NSLog(@"Failed to create folder %@. Error = %@",
          paramPath,
          error);
  }
}

/* Создаем пять файлов с расширением .txt в заданном каталоге, называем
  их 1.txt, 2.txt и т. д. */
- (void) createFilesInFolder:(NSString *)paramPath{

  /* Создаем 10 файлов */
  for (NSUInteger counter = 0; counter < 5; counter++){
    NSString *fileName = [NSString stringWithFormat:@"%lu.txt",
                          (unsigned long)counter+1];
    NSString *path = [paramPath stringByAppendingPathComponent:fileName];
    NSString *fileContents = [NSString stringWithFormat:@"Some text"];
    NSError *error = nil;
    if ([fileContents writeToFile:path
                       atomically:YES
                         encoding:NSUTF8StringEncoding
                            error:&error] == NO){
      NSLog(@"Failed to save file to %@. Error = %@", path, error);
    }
  }

}

/* Перечисляем все файлы/каталоги, расположенные по заданному пути */
- (void) enumerateFilesInFolder:(NSString *)paramPath{

  NSError *error = nil;
  NSArray *contents = [self.fileManager contentsOfDirectoryAtPath:paramPath
                                                            error:&error];

  if ([contents count] > 0 &&
    error == nil){
    NSLog(@"Contents of path %@ = \n%@", paramPath, contents);
  }
  else if ([contents count] == 0 &&
    error == nil){
```

```
    NSLog(@"Contents of path %@ is empty!", paramPath);
  }
  else {
    NSLog(@"Failed to enumerate path %@. Error = %@", paramPath, error);
  }

}

/* Удаляем все файлы/каталоги по заданному пути */
- (void) deleteFilesInFolder:(NSString *)paramPath{

  NSError *error = nil;
  NSArray *contents = [self.fileManager contentsOfDirectoryAtPath:paramPath
                                                            error:&error];

  if (error == nil){
    error = nil;
    for (NSString *fileName in contents){
      /* У нас есть имя файла, но, чтобы удалить этот файл,
      нужен полный путь к нему */
      NSString *filePath = [paramPath
                            stringByAppendingPathComponent:fileName];
      if ([self.fileManager removeItemAtPath:filePath
           error:&error] == NO){
        NSLog(@"Failed to remove item at path %@. Error = %@",
              fileName,
              error);
      }
    }
  } else {
    NSLog(@"Failed to enumerate path %@. Error = %@", paramPath, error);
  }

}

/* Удаляем каталог, к которому ведет заданный путь*/
- (void) deleteFolder:(NSString *)paramPath{
  NSError *error = nil;
  if ([self.fileManager removeItemAtPath:paramPath error:&error] == NO){
    NSLog(@"Failed to remove path %@. Error = %@", paramPath, error);
  }
}
```

Не забывайте: свойство `fileManager`, которое мы используем в различных методах делегата нашего приложения, — это свойство самого делегата приложения, определяемое следующим образом:

```
#import "AppDelegate.h"

@interface AppDelegate ()
```

```
@property (nonatomic, strong) NSFileManager *fileManager;
@end

@implementation AppDelegate

<# Здесь находится остаток кода делегата приложения #>
```

В коде из этого примера объединено немало концепций, изученных в этой главе, — от перечисления до создания и удаления файлов. Все это вы здесь найдете. Как видите, с момента начала разработки приложения мы выполняем шесть основных задач, для каждой из которых существуют собственные методы.

1. Создание каталога `tmp/txt`. Мы знаем, что каталог `tmp` создается в iOS для каждого приложения, но подкаталог `txt` на момент установки приложения отсутствует.
2. Создание пяти файлов в каталоге `tmp/txt`.
3. Перечисление всех файлов в каталоге `tmp/txt`. Эту операцию нужно выполнить лишь для того, чтобы убедиться, что мы успешно создали в этом каталоге все пять файлов.
4. Удаление всех созданных файлов — собственно, именно эта операция интересовала нас в данном разделе.
5. Повторное перечисление файлов в каталоге `tmp/txt`. Эту операцию мы выполняем для того, чтобы убедиться, что механизм удаления сработал правильно.
6. Удаление каталога `tmp/txt`, ведь он нам больше не нужен. Повторюсь: обязательно учитывайте, какие файлы и каталоги вы создаете на диске. Дисковое пространство на дороге не валяется! Поэтому, если какие-то файлы или каталоги вам больше не нужны, обязательно их удаляйте.

## См. также

Раздел 12.2.

# 12.6. Сохранение объектов в файлах

## Постановка задачи

Вы добавили в ваш проект новый класс и теперь хотите сохранить этот объект на диск в виде файла, а потом в случае необходимости считать этот файл с диска.

## Решение

Убедитесь, что ваш класс соответствует протоколу `NSCoding`, и реализуйте все необходимые методы данного протокола. Не волнуйтесь, я все подробно объясню в подразделе «Обсуждение» данного раздела.

## Обсуждение

В iOS SDK есть два очень удобных класса, предназначенных именно для этой цели. Процесс, который будет описан в этом разделе, в программировании называется *«маршалинг»*. Вот эти классы.

- `NSKeyedArchiver` — класс, позволяющий архивировать или сохранять содержимое объекта или дерева объектов по ключам. Каждое значение в классе, скажем каждое свойство, может быть сохранено в архиве с применением ключа, выбранного программистом. Вы получаете архивный файл (далее мы обсудим этот момент подробнее) и просто сохраняете ваши значения с ключами, которые выбраны вами же. Точно как в словаре!
- `NSKeyedUnarchiver` — этот класс работает совершенно противоположным образом. Он просто дает вам неархивированный словарь и предлагает считать значения в свойства вашего объекта.

Для обеспечения работы класса-архиватора и класса-деархиватора необходимо гарантировать, что те объекты, архивацию и деархивацию которых вы запрашиваете, соответствуют протоколу `NSCoding`. Начнем с простого класса `Person`. Вот заголовок этого класса:

```
#import <Foundation/Foundation.h>
@interface Person : NSObject <NSCoding>

@property (nonatomic, copy) NSString *firstName;
@property (nonatomic, copy) NSString *lastName;

@end
```

Теперь, если вы не напишете никакого кода реализации для этого класса и попытаетесь скомпилировать имеющийся код, то компилятор начнет забрасывать вас предупреждениями, сводящимися к следующему: класс не соответствует протоколу `NSCoding` и не реализует необходимые методы этого протокола. Вот какие методы нам потребуется реализовать:

- `- (void)encodeWithCoder:(NSCoder *)aCoder` — от этого метода мы получаем сущность-кодировщик. Кодировщик используется точно так же, как словарь. Просто храните в нем значения с ключами на ваш выбор;
- `- (instancetype)initWithCoder:(NSCoder *)aDecoder;` — этот метод вызывается в вашем классе всякий раз, когда вы пытаетесь разархивировать класс с помощью `NSKeyedUnarchiver`. Просто считывайте значения их экземпляра `NSCoder`, передаваемого этому методу.

Итак, учитывая сказанное, реализуем наш класс:

```
#import "Person.h"

NSString *const kFirstNameKey = @"FirstNameKey";
NSString *const kLastNameKey = @"LastNameKey";

@implementation Person
```

```
- (void)encodeWithCoder:(NSCoder *)aCoder{
  [aCoder encodeObject:self.firstName forKey:kFirstNameKey];
  [aCoder encodeObject:self.lastName forKey:kLastNameKey];
}

- (instancetype)initWithCoder:(NSCoder *)aDecoder{
  self = [super init];
  if (self != nil){
    _firstName = [aDecoder decodeObjectForKey:kFirstNameKey];
    _lastName = [aDecoder decodeObjectForKey:kLastNameKey];
  }
  return self;
}

@end
```

Как видите, мы работаем с экземпляром класса `NSCoder` практически так же, как и со словарем. Разница заключается в том, что вместо словарного метода `setValue:forKey:` мы пользуемся `encodeObject:forKey:`, а вместо словарного метода `objectForKey:` задействуем `decodeObjectForKey:`. Отличия от словарей минимальны.

Итак, с этим классом все понятно. Теперь реализуем механизм архивации и деархивации, пользуясь двумя вышеупомянутыми классами. Мы собираемся сначала инстанцировать объект типа `Person`, затем заархивировать его, убрать из памяти, потом считать обратно из файла и убедиться, что разархивированное значение совпадает с тем, которое мы изначально записали в класс. Все это мы реализуем в делегате приложения, поскольку там это будет сделать проще всего:

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

@implementation AppDelegate

- (BOOL) application:(UIApplication *)application
didFinishLaunchingWithOptions:(NSDictionary *)launchOptions{

  /* Определяем имя и фамилию, которые собираемся задать в объекте */
  NSString *const kFirstName = @"Steven";
  NSString *const kLastName = @"Jobs";

  /* Определяем, где хотим заархивировать объект */
  NSString *filePath = [NSTemporaryDirectory()
  stringByAppendingPathComponent:@"steveJobs.txt"];

  /* Инстанцируем объект */
  Person *steveJobs = [[Person alloc] init];
  steveJobs.firstName = kFirstName;
  steveJobs.lastName = kLastName;

  /* Архивируем объект в файл */
```

```
  [NSKeyedArchiver archiveRootObject:steveJobs toFile:filePath];

  /* Теперь разархивируем этот же класс в другой объект */
  Person *cloneOfSteveJobs =
  [NSKeyedUnarchiver unarchiveObjectWithFile:filePath];

  /* Проверяем, совпадают ли имя и фамилия в разархивированном объекте
   с именем и фамилией, которые находились в ранее архивированном объекте */
  if ([cloneOfSteveJobs.firstName isEqualToString:kFirstName] &&
    [cloneOfSteveJobs.lastName isEqualToString:kLastName]){
    NSLog(@"Unarchiving worked");
  } else {
    NSLog(@"Could not read the same values back. Oh no!");
  }

  /* Временный файл нам больше не нужен, удаляем его */
  NSFileManager *fileManager = [[NSFileManager alloc] init];
    [fileManager removeItemAtPath:filePath error:nil];

  self.window = [[UIWindow alloc]
                 initWithFrame:[[UIScreen mainScreen] bounds]];
  self.window.backgroundColor = [UIColor whiteColor];
  [self.window makeKeyAndVisible];
  return YES;
}
```

Итак, при архивации просто используется метод класса `archiveRootObject:toFile`, относящийся к классу `NSKeyedArchiver`. Этот метод принимает объект и файл, в котором должно быть сохранено содержимое. Все просто. А если нужно разархивировать информацию? Не сложнее архивации. Нам просто нужно найти путь к заархивированному файлу, передать его методу класса `unarchiveObjectWithFile:`, относящемуся к классу `NSKeyedUnarchiver`. Всю остальную работу класс выполнит за вас.

## См. также

Раздел 12.1.

# 13 Камера и библиотека фотографий

## 13.0. Введение

Большинство устройств с операционной системой iOS, допустим iPhone, оборудованы камерами. У самого нового iPhone две камеры, у других моделей может быть всего по одной. Некоторые устройства с операционной системой iOS не оснащены камерами. Класс `UIImagePickerController` позволяет программисту отображать для пользователя привычный интерфейс Camera и предлагать сделать снимок или записать видео. Фотографии или видеозаписи, выполненные с помощью класса `UIImagePickerController`, становятся доступны программисту.

В этой главе будет рассказано, как обеспечить пользователю возможность снимать фотографии и записывать видео прямо из приложения, получать доступ к фотографиям и видео, размещенным в библиотеке фотографий (Photo Library) на устройстве с iOS, например на iPod touch или iPad.

В симуляторе iOS интерфейс Camera не поддерживается. Все приложения, в которых требуется применять этот интерфейс, следует тестировать и отлаживать на настоящем устройстве с iOS, которое оборудовано камерой.

В этой главе мы сначала попытаемся определить, имеется ли камера на том устройстве с iOS, где используется наше приложение. Кроме того, вы можете выяснить, позволяет ли камера вам (программисту) делать фотоснимки, записывать видео или доступны обе эти функции. Для этого необходимо добавить фреймворк `MobileCoreServices.framework` к целевой сборке. Просто импортируйте его обобщающий фреймворк в ваше приложение, вот так:

```
#import "AppDelegate.h"
#import <MobileCoreServices/MobileCoreServices.h>

@implementation AppDelegate

<# Остаток вашего кода находится здесь #>
```

Далее перейдем к изучению других тем, в частности рассмотрим доступ к видео и фотографиям, расположенным в различных альбомах на устройстве с iOS. Речь идет о тех же самых альбомах, в которые можно попасть через приложение Photos (Фотографии), интегрированное в операционную систему iOS.

Но получить доступ к фотографии, находящейся в альбоме, проще, чем к видеозаписи. При работе с фотографиями мы получим адрес снимка и сможем просто загрузить эту информацию об изображении в экземпляр `NSData` либо прямо в экземпляр `UIImage`. В аналогичном случае с видео мы не получим адрес, по которому файл находится в файловой системе и с которого можно загрузить нужное видео. Вместо этого получим примерно такой адрес:

```
assets-library://asset/asset.MOV?id=10000000004&ext=MOV
```

При работе с подобными адресами необходимо использовать фреймворк Assets Library (Библиотека ресурсов). Библиотека ресурсов открывает нам доступ к контенту, который обычно предоставляется через приложение Photos (Фотографии). Это, например, фотографии и видеоролики, отснятые пользователем. Кроме того, библиотека ресурсов может применяться для сохранения изображений и видео на устройстве. Потом эти фотографии и ролики будут доступны для библиотеки фотографий (Photo Library), а также других приложений, которым требуется доступ к этому контенту.

Чтобы все коды из этой главы правильно компилировались, убедитесь, что фреймворки Assets Library и Mobile Core Services включены во все ваши файлы с исходным кодом. Для этого можно импортировать заголовочные файлы в файлы с исходным кодом. Предполагается, что вы работаете с последней версией компилятора LLVM, в котором поддерживается работа с модулями:

```
#import "AppDelegate.h"
#import <MobileCoreServices/MobileCoreServices.h>
#import <AssetsLibrary/AssetsLibrary.h>

@implementation AppDelegate

<# Остаток вашего кода находится здесь #>
```

Чтобы все коды из этой главы правильно компилировались, выполните следующие шаги — так вы добавите в ваш проект фреймворк Assets Library.

1. В Xcode щелкните на ярлыке проекта.
2. Выберите цель, к которой вы хотите добавить фреймворк.
3. В верхней части интерфейса выберите Build Phases (Этапы сборки).
4. Нажмите кнопку + в нижнем левом углу раздела Link Binaries with Libraries (Связать двоичные файлы с библиотеками).
5. Выберите из списка фреймворк MobileCoreServices.framework.
6. Нажмите Add (Добавить).

Чтобы получить доступ к данным ресурса, имея ссылку на этот ресурс, выполните следующие шаги.

1. Выделите и инициализируйте объект типа `ALAssetsLibrary`. Объект из библиотеки ресурсов предоставляет специальную перемычку (Bridge), обеспечивающую

доступ к тем видеороликам и фотографиям, которые доступны для приложения Photos (Фотографии).

2. Для доступа к ресурсу воспользуйтесь методом экземпляра `assetForURL:resultBlock:failureBlock`, относящимся к объекту библиотеки ресурсов (выделение и инициализация этого объекта были выполнены на шаге 1). Ресурс может представлять собой изображение, видео или любой другой объект, который Apple потенциально может добавить в библиотеку фотографий. Этот метод работает с блоковыми объектами. Подробнее о блоковых объектах и GCD рассказано в главе 7.
3. Высвободите тот объект библиотеки ресурсов, который был выделен и инициализирован на шаге 1.

На этом этапе у вас может возникнуть вопрос: как же именно я получаю доступ к данным ресурса? Параметр `resultBlock` метода экземпляра `assetForURL:resultBlock:failureBlock`, относящегося к объекту библиотеки ресурсов, должен указывать на блоковый объект, принимающий единственный параметр типа `ALAsset`. `ALAsset` — это класс, предоставляемый в библиотеке ресурсов, он инкапсулирует (включает в себя) ресурс, доступный для Photos (Фотографии) или любого другого приложения iOS, пытающегося использовать этот ресурс. Тема сохранения фотоснимков и видеороликов в библиотеке фотографий более подробно рассмотрена в разделах 13.4 и 13.5. О получении фотографий и видео из библиотеки фотографий и библиотеки ресурсов подробнее рассказано в разделах 13.6 и 13.7.

# 13.1. Обнаружение и испытание камеры

## Постановка задачи

Требуется узнать, есть ли камера на том устройстве с iOS, где работает ваше приложение, и можете ли вы получить доступ к этой камере. Это очень важный момент, который нужно проверить, прежде чем приступать к работе с камерой. Ведь нельзя исключить вероятность того, что ваше приложение будет использоваться и на каких-то устройствах, не оснащенных камерой.

## Решение

Применяйте метод класса `isSourceTypeAvailable:`, относящийся к классу `UIImagePickerController`, со значением `UIImagePickerControllerSourceTypeCamera` следующим образом:

```
- (BOOL) isCameraAvailable{

  return [UIImagePickerController isSourceTypeAvailable:
          UIImagePickerControllerSourceTypeCamera];

}

- (BOOL)            application:(UIApplication *)application
```

```
  didFinishLaunchingWithOptions:(NSDictionary *)launchOptions{

  if ([self isCameraAvailable]){
    NSLog(@"Camera is available.");
  } else {
    NSLog(@"Camera is not available.");
  }

  self.window = [[UIWindow alloc] initWithFrame:
                 [[UIScreen mainScreen] bounds]];

  self.window.backgroundColor = [UIColor whiteColor];
  [self.window makeKeyAndVisible];
  return YES;
}
```

## Обсуждение

Прежде чем попытаться отобразить для пользователя экземпляр `UIImagePickerController`, позволяющий делать фотоснимки или записывать видео, нужно проверить, поддерживается ли на устройстве этот интерфейс. Метод класса `isSourceTypeAvailable:` позволяет определить три источника данных:

- камеру — для этого данному методу сообщается значение `UIImagePickerControllerSourceTypeCamera`;
- библиотеку фотографий — для этого данному методу сообщается значение `UIImagePickerControllerSourceTypePhotoLibrary`. В результате включается обзор корневого каталога в директории `Photos` на устройстве;
- каталог с фотографиями, отснятыми с камеры данного устройства (Camera Roll), — в таком случае метод получает значение `UIImagePickerControllerSourceTypeSavedPhotosAlbum`.

Если вы собираетесь проверить доступность любой из этих функций на устройстве с iOS, нужно передать описанные значения методу класса `isSourceTypeAvailable:`, относящемуся к классу `UIImagePickerController`, и лишь потом попробовать отобразить для пользователя соответствующий интерфейс.

Теперь можно воспользоваться методами класса `isSourceTypeAvailable:` и `availableMediaTypesForSourceType:`, относящимися к классу `UIImagePickerController`, чтобы для начала определить, доступен ли источник медийной информации (например, камера, библиотека фотографий и т. д.). Если источник имеется, определим, какие типы медиаинформации (например, изображения или видео) в нем предоставляются:

```
- (BOOL) cameraSupportsMedia:(NSString *)paramMediaType
            sourceType:(UIImagePickerControllerSourceType)paramSourceType{

  __block BOOL result = NO;

  if ([paramMediaType length] == 0){
```

```
    NSLog(@"Media type is empty.");
    return NO;
  }

  NSArray *availableMediaTypes =
  [UIImagePickerController
          availableMediaTypesForSourceType:paramSourceType];

  [availableMediaTypes enumerateObjectsUsingBlock:
   ^(id obj, NSUInteger idx, BOOL *stop) {

     NSString *mediaType = (NSString *)obj;
     if ([mediaType isEqualToString:paramMediaType]){
       result = YES;
       *stop= YES;
     }

   }];

  return result;

}

- (BOOL) doesCameraSupportShootingVideos{

  return [self cameraSupportsMedia:(__bridge NSString *)kUTTypeMovie
                        sourceType:UIImagePickerControllerSourceTypeCamera];

}

- (BOOL) doesCameraSupportTakingPhotos{

  return [self cameraSupportsMedia:(__bridge NSString *)kUTTypeImage
                        sourceType:UIImagePickerControllerSourceTypeCamera];

}

- (BOOL)            application:(UIApplication *)application
  didFinishLaunchingWithOptions:(NSDictionary *)launchOptions{

  if ([self doesCameraSupportTakingPhotos]){
    NSLog(@"The camera supports taking photos.");
  } else {
    NSLog(@"The camera does not support taking photos");
  }

  if ([self doesCameraSupportShootingVideos]){
    NSLog(@"The camera supports shooting videos.");
  } else {
    NSLog(@"The camera does not support shooting videos.");
```

```
    }

    self.window = [[UIWindow alloc] initWithFrame:
                   [[UIScreen mainScreen] bounds]];

    self.window.backgroundColor = [UIColor whiteColor];
    [self.window makeKeyAndVisible];
    return YES;
}
```

Мы приводим типы значений kUTTypeMovie и kUTTypeImage к NSString с помощью __bridge (как было рассказано в разделе 1.18). Это объясняется тем, что два вышеупомянутых значения относятся к типу CFStringRef и нам нужно получить их представление в виде NSString. Чтобы упростить работу статического анализатора и компилятора и не получать от компилятора лишних сообщений, лучше выполнить такое приведение типов.

На некоторых устройствах с iOS может быть установлена не одна камера. Например, их может быть две — передняя и задняя. Чтобы определить, доступны ли эти камеры, воспользуйтесь методом класса `isCameraDeviceAvailable:`, относящимся к классу `UIImagePickerController`:

```
- (BOOL) isFrontCameraAvailable{

    return [UIImagePickerController
            isCameraDeviceAvailable:UIImagePickerControllerCameraDeviceFront];

}

- (BOOL) isRearCameraAvailable{

    return [UIImagePickerController
            isCameraDeviceAvailable:UIImagePickerControllerCameraDeviceRear];

}
```

Если вызвать эти методы на не самом новом iPhone, где отсутствует задняя камера, то можно заметить, что метод `isFrontCameraAvailable` возвращает `NO`, а метод `isRearCameraAvailable` — `YES`. При запуске данного кода на iPhone, оснащенном как передней, так и задней камерами, оба метода вернут `YES`, поскольку на iPhone 4 имеются две камеры — спереди и сзади.

Если в вашем приложении недостаточно просто определить, какая камера имеется на устройстве, можно получить и другие настройки, воспользовавшись классом `UIImagePickerController`. Одна из этих настроек позволяет узнать, есть ли на камере данного устройства функция вспышки. Метод класса `isFlashAvailableForCameraDevice:`, относящийся к классу `UIImagePickerController`, применяется, чтобы выяснить, на какой камере доступна функция вспышки — передней или задней. Не забывайте также, что метод класса `isFlashAvailableForCameraDevice:`, относящийся к классу `UIImagePickerController`, сначала проверяет доступность запрошенной камеры, а уже потом проверяется доступность функции вспышки на этой камере.

Поэтому методы, которые мы здесь реализуем, можно будет запускать и на устройствах, лишенных передней или задней камеры, без необходимости предварительной проверки доступности камеры:

```
- (BOOL) isFlashAvailableOnFrontCamera{

  return [UIImagePickerController isFlashAvailableForCameraDevice:
          UIImagePickerControllerCameraDeviceFront];

}

- (BOOL) isFlashAvailableOnRearCamera{

  return [UIImagePickerController isFlashAvailableForCameraDevice:
          UIImagePickerControllerCameraDeviceRear];

}
```

Теперь, если воспользоваться всеми методами, написанными в этом разделе, и протестировать их, например, в делегате нашего приложения, мы сможем увидеть результаты на различных устройствах:

```
- (BOOL)            application:(UIApplication *)application
  didFinishLaunchingWithOptions:(NSDictionary *)launchOptions{

  if ([self isFrontCameraAvailable]){
    NSLog(@"The front camera is available.");
    if ([self isFlashAvailableOnFrontCamera]){
      NSLog(@"The front camera is equipped with a flash");
    } else {
      NSLog(@"The front camera is not equipped with a flash");
    }
  } else {
    NSLog(@"The front camera is not available.");
  }

  if ([self isRearCameraAvailable]){
    NSLog(@"The rear camera is available.");
    if ([self isFlashAvailableOnRearCamera]){
      NSLog(@"The rear camera is equipped with a flash");
    } else {
      NSLog(@"The rear camera is not equipped with a flash");
    }
  } else {
    NSLog(@"The rear camera is not available.");
  }

  if ([self doesCameraSupportTakingPhotos]){
    NSLog(@"The camera supports taking photos.");
  } else {
    NSLog(@"The camera does not support taking photos");
```

```
  }

  if ([self doesCameraSupportShootingVideos]){
    NSLog(@"The camera supports shooting videos.");
  } else {
    NSLog(@"The camera does not support shooting videos.");
  }

  self.window = [[UIWindow alloc] initWithFrame:
                 [[UIScreen mainScreen] bounds]];

  self.window.backgroundColor = [UIColor whiteColor];
  [self.window makeKeyAndVisible];
  return YES;
}
```

Вот результаты запуска данного приложения на новом iPhone:

```
The front camera is available.                // передняя камера доступна
The front camera is not equipped with a flash // передняя камера
                                              // не оснащена функцией вспышки
The rear camera is available.              // задняя камера доступна
The rear camera is equipped with a flash   // задняя камера оснащена
,                                          // функцией вспышки
The camera supports taking photos.         // камера позволяет делать
,                                          // фотоснимки
The camera supports shooting videos.       // камера позволяет
                                           // записывать видео
```

Вот вывод того же кода при запуске на симуляторе iPhone:

```
The front camera is not available.  // передняя камера недоступна
The rear camera is not available.   // задняя камера недоступна
The camera does not support taking photos // камера не поддерживает съемку фотографий
The camera does not support shooting videos // камера не поддерживает съемку видео
```

# 13.2. Фотографирование с помощью камеры

## Постановка задачи

Требуется попросить пользователя сделать снимок (фотография выполняется камерой устройства). После того как пользователь сделает снимок, необходимо получить доступ к этой фотографии.

## Решение

Инстанцируйте объект типа `UIImagePickerController` и представьте его пользователю как модальный вид в актуальном контроллере вида. Вот объявление этого контроллера вида:

```
#import "ViewController.h"
#import <MobileCoreServices/MobileCoreServices.h>

@interface ViewController ()<UIImagePickerControllerDelegate,
UINavigationControllerDelegate>

@end

@implementation ViewController

<# Остаток вашего кода находится здесь #>
```

Делегат экземпляра `UIImagePickerController` должен соответствовать протоколам `UINavigationControllerDelegate` и `UIImagePickerControllerDelegate`. Если вы забудете включить их в `.h`-файл вашего объекта-делегата, то будете получать предупреждения от компилятора при присвоении значения делегатному свойству контроллера для выбора изображений. Не забывайте, что вы можете присваивать делегату объект экземпляра `UIImagePickerController` и в том случае, если данный объект не соответствует явно протоколам `UIImagePickerControllerDelegate` и `UINavigationControllerDelegate`, но реализует методы, необходимые в этих протоколах. Тем не менее я советую «подсказывать» компилятору, что фактически объект-делегат соответствует вышеупомянутым протоколам, — так вы избавитесь от лишних предупреждений компилятора.

В реализации контроллера вида попытаемся отобразить контроллер для выбора изображения в виде модального контроллера вида следующим образом:

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

  static BOOL beenHereBefore = NO;

  if (beenHereBefore){
    /* Отображаем элемент для выбора даты только после того, как вызывается
     метод viewDidAppear:, что происходит при каждом отображении вида
     нашего контроллера вида */
    return;
  } else {
    beenHereBefore = YES;
  }

  if ([self isCameraAvailable] &&
      [self doesCameraSupportTakingPhotos]){

    UIImagePickerController *controller =
      [[UIImagePickerController alloc] init];

    controller.sourceType = UIImagePickerControllerSourceTypeCamera;

    NSString *requiredMediaType = (__bridge NSString *)kUTTypeImage;
```

```
    controller.mediaTypes = [[NSArray alloc]
                             initWithObjects:requiredMediaType, nil];

    controller.allowsEditing = YES;
    controller.delegate = self;

    [self.navigationController presentModalViewController:controller
                                                 animated:YES];

  } else {
    NSLog(@"Camera is not available.");
  }
}
```

В этом примере пользуемся методами isCameraAvailable и doesCameraSupportTakingPhotos. Эти методы реализованы и подробно рассмотрены в разделе 13.1.

В данном примере мы предоставим пользователю возможность делать снимки, пользуясь контроллером для выбора изображений. Вы, должно быть, заметили, что для делегатного свойства инструмента выбора изображений мы задаем значение `self`, которое относится к контроллеру вида. При этом необходимо убедиться, что мы реализовали методы, определенные в протоколе `UIImagePickerControllerDelegate`:

```
- (void)  imagePickerController:(UIImagePickerController *)picker
  didFinishPickingMediaWithInfo:(NSDictionary *)info{

  NSLog(@"Picker returned successfully.");
NSLog(@"%@", info);

  NSString     *mediaType = [info objectForKey:
                             UIImagePickerControllerMediaType];

  if ([mediaType isEqualToString:(__bridge NSString *)kUTTypeMovie]){

    NSURL *urlOfVideo =
    [info objectForKey:UIImagePickerControllerMediaURL];
     NSLog(@"Video URL = %@", urlOfVideo);

  }

  else if ([mediaType isEqualToString:(__bridge NSString *)kUTTypeImage]){
    /* Получим метаданные. Это касается
       только изображений, но не видеороликов. */

    NSDictionary *metadata =
    [info objectForKey:
     UIImagePickerControllerMediaMetadata];

    UIImage *theImage =
```

```
    [info objectForKey:
     UIImagePickerControllerOriginalImage];

     NSLog(@"Image Metadata = %@", metadata);
     NSLog(@"Image = %@", theImage);

  }

  [picker dismissModalViewControllerAnimated:YES];

}

- (void)imagePickerControllerDidCancel:(UIImagePickerController *)picker{

  NSLog(@"Picker was cancelled");
  [picker dismissModalViewControllerAnimated:YES];

}
```

## Обсуждение

При работе с делегатом инструмента для выбора изображений необходимо учитывать пару немаловажных аспектов. Два делегатных метода вызываются у объекта-делегата контроллера для выбора изображений. Метод `imagePickerController:didFinishPickingMediaWithInfo:` вызывается, когда пользователь завершает работу с инструментом выбора изображений (то есть делает снимок и наконец нажимает кнопку). В свою очередь, метод `imagePickerControllerDidCancel:` вызывается в случае, когда операция с инструментом выбора изображений отменяется.

Кроме того, метод делегата `imagePickerController:didFinishPickingMediaWithInfo:` содержит данные о том предмете, который был отснят пользователем, независимо от того, изображение это или видеоданные. Параметр `didFinishPickingMediaWithInfo` — это словарь значений, сообщающий, что именно было отснято в инструменте выбора изображений плюс метаданные отснятого контента и другую полезную информацию. Работу в этом методе следует начать со считывания значения ключа `UIImagePickerControllerMediaType` из этого словаря. Объект для данного ключа представляет собой экземпляр `NSString` и может принимать одно из следующих значений:

- `kUTTypeImage` — фотография, сделанная камерой;
- `kUTTypeMovie` — видеоролик, отснятый камерой.

Значения kUTTypeImage и kUTTypeMovie доступны во фреймворке Mobile Core Services и относятся к типу CFStringRef. При необходимости можно просто привести тип этих значений к NSString.

Определив тип ресурса, созданного камерой (то есть узнав, идет речь о фотографии или видеоролике), можно получить доступ к свойствам этого ресурса, вновь воспользовавшись параметром словаря `didFinishPickingMediaWithInfo`.

При работе с изображениями (kUTTypeImage) можно получить доступ к следующим ключам:

- UIImagePickerControllerMediaMetadata — объектом, хранимым по этому ключу, является объект типа NSDictionary. В этом словаре содержится масса полезной информации об изображении, отснятом пользователем. Подробное обсуждение значений, содержащихся в этом словаре, выходит за рамки этой главы;
- UIImagePickerControllerOriginalImage — объектом, хранимым по этому ключу, является объект типа UIImage. В нем содержится изображение, отснятое пользователем;
- UIImagePickerControllerCropRect — если вы активизировали возможность редактирования (с помощью свойства allowsEditing объекта UIImagePickerController), то в объекте этого ключа будет содержаться прямоугольник, по которому была сделана обрезка;
- UIImagePickerControllerEditedImage — если вы активизировали возможность редактирования (с помощью свойства allowsEditing объекта UIImagePickerController), то в значении этого ключа будет содержаться отредактированное изображение (масштабированное, с измененными размерами).

Для видеороликов (kUTTypeMovie), отснятых пользователем, можно получать доступ к ключу UIImagePickerControllerMediaURL в параметре словаря didFinishPickingMediaWithInfo в методе imagePickerController:didFinishPickingMediaWithInfo:. Значение этого ключа представляет собой объект типа NSURL, содержащий URL видеоролика, отснятого пользователем.

После того как вы получите ссылку на экземпляр UIImage, отснятый пользователем с помощью камеры, можете просто начинать использовать этот экземпляр в своем приложении.

Фотографии, снятые внутри приложения с помощью инструмента для выбора изображений, по умолчанию не сохраняются в каталоге для снимков фотокамеры (Camera Roll).

## См. также

Раздел 13.1.

# 13.3. Запись видео с помощью камеры

## Постановка задачи

Необходимо обеспечить пользователю возможность записи видео со своего устройства с системой iOS. Кроме того, вы сами должны иметь возможность применять это видео в своем приложении.

## Решение

Воспользуйтесь объектом UIImagePickerController с источником типа UIImagePickerControllerSourceTypeCamera и медийной информацией типа kUTTypeMovie:

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

  static BOOL beenHereBefore = NO;

  if (beenHereBefore){
    /* Отображаем элемент для выбора даты только после того, как вызывается
    метод viewDidAppear:, что происходит при каждом отображении вида
    нашего контроллера вида */
    return;
  } else {
    beenHereBefore = YES;
  }
  if ([self isCameraAvailable] &&
      [self doesCameraSupportTakingPhotos]){

    UIImagePickerController *controller =
      [[UIImagePickerController alloc] init];

    controller.sourceType = UIImagePickerControllerSourceTypeCamera;

    controller.mediaTypes = @[(__bridge NSString *)kUTTypeMovie];
    controller.allowsEditing = YES;
    controller.delegate = self;

    [self.navigationController presentModalViewController:controller
                                                 animated:YES];

  } else {
    NSLog(@"Camera is not available.");
  }

}
```

Методы isCameraAvailable и doesCameraSupportShootingVideos, использованные в данном примере, реализованы и обсуждены в разделе 13.1.

Вот как мы реализуем методы делегата инструмента для выбора изображений:

```
- (void)  imagePickerController:(UIImagePickerController *)picker
  didFinishPickingMediaWithInfo:(NSDictionary *)info{

  NSLog(@"Picker returned successfully.");

  NSLog(@"%@", info);

  NSString     *mediaType = [info objectForKey:
                             UIImagePickerControllerMediaType];

  if ([mediaType isEqualToString:(__bridge NSString *)kUTTypeMovie]){
```

```
    NSURL *urlOfVideo =
    [info objectForKey:UIImagePickerControllerMediaURL];

    NSLog(@"Video URL = %@", urlOfVideo);

    NSError *dataReadingError = nil;

    NSData *videoData =
    [NSData dataWithContentsOfURL:urlOfVideo
                          options:NSDataReadingMapped
                            error:&dataReadingError];

    if (videoData != nil){
      /* Нам удалось считать данные. */
      NSLog(@"Successfully loaded the data.");
    } else {
      /* Нам не удалось считать данные. Используем переменную
         dataReadingError, чтобы определить, в чем заключается ошибка. */
      NSLog(@"Failed to load the data with error = %@",
            dataReadingError);
    }

  }
  [picker dismissModalViewControllerAnimated:YES];

}

- (void)imagePickerControllerDidCancel:(UIImagePickerController *)picker{

  NSLog(@"Picker was cancelled");
  [picker dismissModalViewControllerAnimated:YES];

}
```

## Обсуждение

Как только вы обнаружите, что устройство с системой iOS, на котором работает ваше приложение, поддерживает запись видео, можно открыть инструмент для выбора изображений с типом источника `UIImagePickerControllerSourceTypeCamera` и типом медийной информации `kUTTypeMovie`, чтобы пользователь вашего приложения мог снимать видео. Как только съемка будет закончена, будет вызван метод делегата `imagePickerController:didFinishPickingMediaWithInfo:` и вы сможете использовать параметр словаря `didFinishPickingMediaWithInfo`, чтобы узнать более подробную информацию об отснятом видео. (Значения, которые могут быть размещены в таком словаре, подробно рассмотрены в разделе 13.2.)

Когда пользователь записывает видео, работая при этом с инструментом для выбора изображений, это видео будет сохраняться во временной папке в пакете вашего приложения, а не в каталоге для снимков камеры на устройстве (то есть не

в Camera Roll). Пример такого URL: file://localhost/private/var/mobile/Applications/<APPID>/tmp/captureT0x104e20.tmp.TQ9UTr/capturedvideo.MOV.

Значение APPID в этом URL будет представлять уникальный идентификатор вашего приложения. Разумеется, оно будет специфичным в каждой конкретной программе.

Как программист вы можете предоставить пользователю возможность не только снимать видео на камеру устройства, но и модифицировать уже отснятый видеоматериал. Можно изменить два важных свойства класса `UIImagePickerController`, чтобы откорректировать стандартный ход записи видео:

- `videoQuality` — характеризует качество записываемого видео. Для данного свойства можно выбрать, например, значение `UIImagePickerControllerQualityTypeHigh` или `UIImagePickerControllerQualityTypeMedium`;
- `videoMaximumDuration` — указывает максимальную длительность видео. Это значение измеряется в секундах.

Например, если мы предоставляем пользователю возможность записывать высококачественное видео длительностью до 30 секунд, то можем просто изменить значения вышеупомянутых свойств экземпляра `UIImagePickerController`:

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

  static BOOL beenHereBefore = NO;

  if (beenHereBefore){
    /* Отображаем элемент для выбора даты только после того, как вызывается
    метод viewDidAppear:, что происходит при каждом отображении вида
    нашего контроллера вида */
    return;
  } else {
    beenHereBefore = YES;
  }

  if ([self isCameraAvailable] &&
      [self doesCameraSupportTakingPhotos]){

    UIImagePickerController *controller =
      [[UIImagePickerController alloc] init];

    controller.sourceType = UIImagePickerControllerSourceTypeCamera;

    controller.mediaTypes = @[(__bridge NSString *)kUTTypeMovie];

    controller.allowsEditing = YES;
    controller.delegate = self;

    /* Записываем видео в высоком качестве. */
    controller.videoQuality = UIImagePickerControllerQualityTypeHigh;
```

```
    /* Позволяем записать только 30 секунд видео. */
    controller.videoMaximumDuration = 30.0f;

    [self.navigationController presentModalViewController:controller
                                                  animated:YES];

  } else {
    NSLog(@"Camera is not available.");
  }

}
```

## См. также

Раздел 13.1.

# 13.4. Сохранение снимков в библиотеке фотографий

## Постановка задачи

Необходимо обеспечить возможность сохранения снимков в пользовательской библиотеке фотографий.

## Решение

Воспользуйтесь процедурой `UIImageWriteToSavedPhotosAlbum`:

```
- (void) imageWasSavedSuccessfully:(UIImage *)paramImage
         didFinishSavingWithError:(NSError *)paramError
                      contextInfo:(void *)paramContextInfo{

  if (paramError == nil){
    NSLog(@"Image was saved successfully.");
  } else {
    NSLog(@"An error happened while saving the image.");
    NSLog(@"Error = %@", paramError);
  }

}

- (void)  imagePickerController:(UIImagePickerController *)picker
  didFinishPickingMediaWithInfo:(NSDictionary *)info{

  NSLog(@"Picker returned successfully.");

  NSLog(@"%@", info);
```

```
  NSString     *mediaType = [info objectForKey:
                             UIImagePickerControllerMediaType];

  if ([mediaType isEqualToString:(__bridge NSString *)kUTTypeImage]){
    UIImage *theImage = nil;

    if ([picker allowsEditing]){
      theImage = [info objectForKey:UIImagePickerControllerEditedImage];
    } else {
      theImage = [info objectForKey:UIImagePickerControllerOriginalImage];
    }

    SEL selectorToCall = @selector(imageWasSavedSuccessfully:
                                   didFinishSavingWithError:contextInfo:);
    UIImageWriteToSavedPhotosAlbum(theImage,
                                   self,
                                   selectorToCall,
                                   NULL);
  }

  [picker dismissModalViewControllerAnimated:YES];

}

- (void)imagePickerControllerDidCancel:(UIImagePickerController *)picker{

  NSLog(@"Picker was cancelled");
  [picker dismissModalViewControllerAnimated:YES];

}

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

  static BOOL beenHereBefore = NO;

  if (beenHereBefore){
    /* Отображаем элемент для выбора даты только после того, как вызывается
     метод viewDidAppear:, что происходит при каждом отображении вида
     нашего контроллера вида */
    return;
  } else {
    beenHereBefore = YES;
  }

  if ([self isCameraAvailable] &&
      [self doesCameraSupportTakingPhotos]){

    UIImagePickerController *controller =
    [[UIImagePickerController alloc] init];
```

```
    controller.sourceType = UIImagePickerControllerSourceTypeCamera;

    NSString *requiredMediaType = (__bridge NSString *)kUTTypeImage;
    controller.mediaTypes = [[NSArray alloc]
                             initWithObjects:requiredMediaType, nil];

    controller.allowsEditing = YES;
    controller.delegate = self;

    [self.navigationController presentModalViewController:controller
                                                 animated:YES];

  } else {
    NSLog(@"Camera is not available.");
  }

}
```

Методы isCameraAvailable и doesCameraSupportTakingPhotos, использованные в данном примере, подробно рассмотрены в разделе 13.1.

## Обсуждение

Обычно после того, как пользователь успешно снимет фотографию на устройство с iOS, он ожидает, что этот снимок сохранится в его библиотеке фотографий. Однако приложения, не входящие в стандартный комплект программ iOS, могут запросить пользователя сделать снимок с помощью класса `UIImagePickerController`, а потом обработать это изображение. В таком случае пользователь поймет, что предоставленное нами приложение может и не сохранять сделанный снимок в библиотеке фотографий, а вместо этого использовать фотографию внутрисистемно. Например, если программа для обмена мгновенными сообщениями позволяет пользователю передавать фотографии на другие устройства, то пользователь поймет, что сделанный им снимок не будет сохранен в библиотеке фотографий, а, напротив, будет передан по Интернету другому пользователю.

Но если вы решите, что хотите сохранить экземпляр `UIImage` в библиотеке фотографий на пользовательском устройстве, то можете применить функцию `UIImageWriteToSavedPhotosAlbum`. Она принимает четыре параметра:

- изображение;
- объект, который станет получать уведомления всякий раз, когда изображение будет полностью сохранено;
- параметр, указывающий селектор (этот селектор будет вызываться применительно к целевому объекту). Данный параметр идет вторым, а селектор вызывается по завершении операции;
- значение контекста, передаваемое указанному селектору, как только операция будет завершена.

Второй, третий и четвертый параметры для этой процедуры указывать не обязательно. Если вы зададите и второй, и третий параметры, то четвертый параметр все равно останется опциональным. Вот, например, селектор, который я выбрал для нашего примера:

```
- (void) imageWasSavedSuccessfully:(UIImage *)paramImage
          didFinishSavingWithError:(NSError *)paramError
                       contextInfo:(void *)paramContextInfo{

  if (paramError == nil){
    NSLog(@"Image was saved successfully.");
  } else {
    NSLog(@"An error happened while saving the image.");
    NSLog(@"Error = %@", paramError);
  }

}
```

Если вы попытаетесь воспользоваться процедурой `UIImageWriteToSavedPhotosAlbum` для сохранения фотоснимка в пользовательской библиотеке фотографий и приложение впервые выполняет эту операцию на данном устройстве, то iOS запросит у пользователя разрешение на такую операцию (рис. 13.1). Таким образом, пользователь может позволить или не позволить приложению сохранять фотографии на устройстве. В конце концов, это пользовательское устройство и мы не можем делать на этом устройстве ничего такого, чего нам не разрешил пользователь. Если пользователь дает разрешение, то процедура `UIImageWriteToSavedPhotosAlbum` продолжит сохранение изображения. Если разрешение не получено, то селектор обработчика завершения по-прежнему будет вызываться, но параметр `didFinishSavingWithError` этого селектора будет устанавливаться в валидный экземпляр ошибки.

Теперь, если пользователь отказывает приложению в праве доступа, все последующие вызовы процедуры `UIImageWriteToSavedPhotosAlbum` будут неуспешными, пока пользователь вручную не изменит настройки своего устройства (рис. 13.2).

Если в данном селекторе вы получаете параметр error, значение которого равно nil, это означает, что изображение было успешно сохранено в пользовательской библиотеке фотографий. В противном случае можно получить значение данного параметра, чтобы выяснить, в чем заключается ошибка.

# 13.5. Сохранение видео в библиотеке фотографий

## Постановка задачи

Требуется сохранить в библиотеке фотографий видеоролик, доступный по URL, например ролик из пакета вашего приложения.


**Рис. 13.1.** iOS запрашивает у пользователя права доступа, прежде чем программа сможет сохранить снимок в библиотеке фотографий

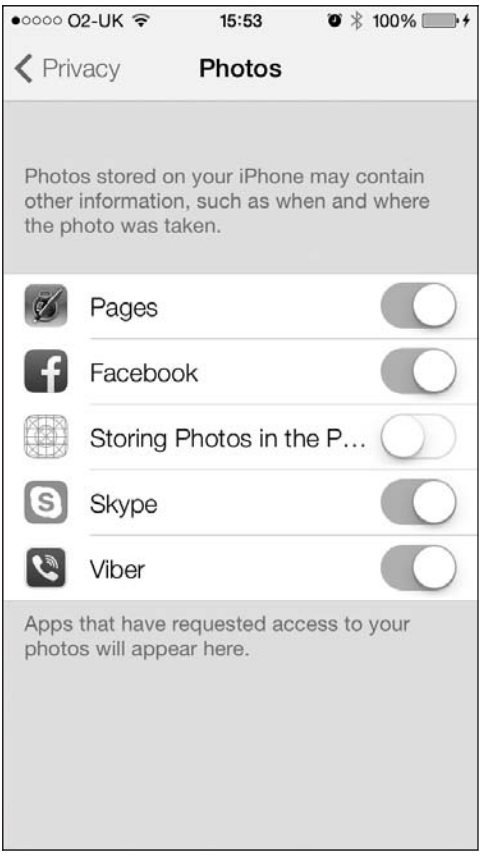


**Рис. 13.2.** Приложению не разрешен доступ к пользовательской библиотеке фотографий

## Решение

Воспользуйтесь методом экземпляра `writeVideoAtPathToSavedPhotosAlbum:completionBlock:`, относящимся к классу `ALAssetsLibrary`:

```
#import "AppDelegate.h"
#import <AssetsLibrary/AssetsLibrary.h>

@interface AppDelegate ()
@property (nonatomic, strong) ALAssetsLibrary *assetsLibrary;
@end

@implementation AppDelegate
- (BOOL)            application:(UIApplication *)application
  didFinishLaunchingWithOptions:(NSDictionary *)launchOptions{
```

```
  self.assetsLibrary = [[ALAssetsLibrary alloc] init];

  NSURL *videoURL = [[NSBundle mainBundle] URLForResource:@"MyVideo"
                                            withExtension:@"MOV"];

  if (videoURL != nil){
    [self.assetsLibrary
     writeVideoAtPathToSavedPhotosAlbum:videoURL
     completionBlock:^(NSURL *assetURL, NSError *error) {

       if (error == nil){
         NSLog(@"no errors happened");
       } else {
         NSLog(@"Error happened while saving the video.");
         NSLog(@"The error is = %@", error);
       }

     }];
  } else {
    NSLog(@"Could not find the video in the app bundle.");
  }

  return YES;
}
```

## Обсуждение

Фреймворк Assets Library — удобный посредник между разработчиком и библиотекой фотографий. Как будет указано в разделе 13.6, в iOS SDK вам предоставляются встроенные компоненты графического пользовательского интерфейса, которыми можно пользоваться для доступа к содержимому библиотеки фотографий. Тем не менее иногда может потребоваться и непосредственный доступ к этому содержимому. В таких случаях следует пользоваться фреймворком библиотеки ресурсов (Assets Library).

Выделив и инициализировав объект Assets Library типа `ALAssetsLibrary`, можно пользоваться методом экземпляра `writeVideoAtPathToSavedPhotosAlbum:completionBlock:`, относящимся к данному объекту, для записи видео с URL в библиотеку фотографий. Все, что от вас требуется, — предоставить URL видеоролика в форме `NSURL`, а также блоковый объект, чей код будет вызываться после сохранения видео. Этот блоковый объект должен принимать два параметра типов `NSURL` и `NSError`.

Если параметр `error` имеет значение `nil`, процесс сохранения прошел нормально и поводов для беспокойства нет. Одна из типичных ошибок, которую iOS может вам вернуть в такой ситуации, примерно такова:

```
Error Domain=ALAssetsLibraryErrorDomain Code=-3302 "Invalid data"
UserInfo=0x7923590 {NSLocalizedFailureReason=
There was a problem writing this asset because
the data is invalid and cannot be viewed or played.,
```

```
NSLocalizedRecoverySuggestion=Try with different data.
NSLocalizedDescription=Invalid data}
```

Такое сообщение об ошибке вы можете получить и в случае, если попытаетесь передать URL (уникальный идентификатор ресурса), не относящийся к пакету вашего приложения.

Первый параметр, передаваемый блоковому объекту, который, в свою очередь, предоставляется методу `writeVideoAtPathToSavedPhotosAlbum:completionBlock:`, будет указывать URL видео, сохраненного в библиотеке ресурсов. URL такого рода может иметь следующий вид:

```
assets-library://asset/asset.MOV?id=F9B5F733-487C-
4418-8C8D-46ABC9FEE23B&ext=MOV
```

Если ваша программа впервые пытается обратиться к библиотеке фотографий на пользовательском устройстве, то iOS спросит у пользователя, разрешена такая операция или нет. Если пользователь даст разрешение, то вызов метода `writeVideoAtPathToSavedPhotosAlbum:completionBlock:` будет успешным. Если же разрешение получено не будет, то объект ошибки внутри блока завершения будет валидным объектом ошибки, который вы можете проверить и над которым можете выполнять действия. Если ранее пользователь не разрешил вашему приложению доступ к библиотеке фотографий, то вы не сможете изменить это решение программно. Лишь когда сам пользователь решит предоставить вам доступ к библиотеке фотографий, он изменит соответствующие настройки в разделах Settings (Настройки) и Privacy (Конфиденциальность).

В разделе 13.7 будет рассказано, как использовать такой URL при загрузке в память данных для видеофайла.

# 13.6. Получение фото и видео из библиотеки фотографий

## Постановка задачи

Необходимо предоставить пользователю возможность выбирать фото или видео из своей библиотеки фотографий и использовать их в приложении.

## Решение

При работе с объектом `UIImagePickerController` используйте `UIImagePickerControllerSourceTypePhotoLibrary` в качестве типа источника и значение `kUTTypeImage` или `kUTTypeMovie` для типа медийной информации:

```
- (BOOL) isPhotoLibraryAvailable{

  return [UIImagePickerController isSourceTypeAvailable:
          UIImagePickerControllerSourceTypePhotoLibrary];
```

```
}

- (BOOL) canUserPickVideosFromPhotoLibrary{

  return [self
          cameraSupportsMedia:(__bridge NSString *)kUTTypeMovie
          sourceType:UIImagePickerControllerSourceTypePhotoLibrary];

}

- (BOOL) canUserPickPhotosFromPhotoLibrary{

  return [self
          cameraSupportsMedia:(__bridge NSString *)kUTTypeImage
          sourceType:UIImagePickerControllerSourceTypePhotoLibrary];

}

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

  static BOOL beenHereBefore = NO;

  if (beenHereBefore){
    /* Отображаем элемент для выбора даты только после того, как вызывается
    метод viewDidAppear:, что происходит при каждом отображении вида
    нашего контроллера вида */
    return;
  } else {
    beenHereBefore = YES;
  }

  if ([self isPhotoLibraryAvailable]){

    UIImagePickerController *controller =
      [[UIImagePickerController alloc] init];

    controller.sourceType = UIImagePickerControllerSourceTypePhotoLibrary;
    NSMutableArray *mediaTypes = [[NSMutableArray alloc] init];

    if ([self canUserPickPhotosFromPhotoLibrary]){
      [mediaTypes addObject:(__bridge NSString *)kUTTypeImage];
    }

    if ([self canUserPickVideosFromPhotoLibrary]){
      [mediaTypes addObject:(__bridge NSString *)kUTTypeMovie];
    }

    controller.mediaTypes = mediaTypes;
    controller.delegate = self;
```

```
    [self.navigationController presentModalViewController:controller
                                                 animated:YES];
  }
}
```

О том, как реализовать метод `cameraSupportsMedia:sourceType:`, который используется в этом примере, рассказано в разделе 13.1.

## Обсуждение

Чтобы пользователь мог выбирать фотоснимки или видеоролики из своей библиотеки фотографий, необходимо установить свойство `sourceType` экземпляра `UIImagePickerController` в значение `UIImagePickerControllerSourceTypePhotoLibrary` и только потом открывать перед пользователем инструмент для выбора изображений. Кроме того, если вы хотите отфильтровать определенные фотографии или видеоролики из общего числа элементов, представленных пользователю в инструменте для выбора изображений, исключите значение `kUTTypeMovie` или `kUTTypeImage` соответственно из списка типов медийной информации, отображаемой в инструменте для выбора изображений (это делается в свойстве `mediaTypes`).

Учитывайте, что, если установить значение свойства `mediaTypes` контроллера для выбора изображений в `nil`, получится пустой массив, который спровоцирует ошибки времени исполнения.

После того как пользователь выберет желаемое изображение, вы будете получать обычные сообщения делегата, соответствующие протоколу `UIImagePickerControllerDelegate`. Подробнее о реализации методов, определяемых в этом протоколе для обработки изображений, рассказано в разделе 13.2.

## См. также

Раздел 13.7.

# 13.7. Получение ресурсов из библиотеки ресурсов

## Постановка задачи

Требуется получить фотографии или видео непосредственно из библиотеки фотографий, не прибегая к использованию каких-либо встроенных компонентов графического пользовательского интерфейса.

## Решение

Воспользуйтесь фреймворком Assets Library (Библиотека ресурсов). Выполните следующие шаги.

1. Выделите и инициализируйте объект типа `ALAssetsLibrary`.
2. Передайте два блоковых объекта методу экземпляра `enumerateGroupsWithTypes:usingBlock:failureBlock:`, относящемуся к объекту «Библиотека ресурсов». Первый блок получит все группы, ассоциированные с типом, который мы сообщили этому методу. Группы будут относиться к типу `ALAssetsGroup`. В случае неудачи этой операции во втором блоке будет возвращена ошибка.
3. Воспользуйтесь методом экземпляра `enumerateAssetsUsingBlock:` каждого группового объекта для перечисления ресурсов, имеющихся в каждой группе. Этот метод принимает единственный параметр — блок, получающий информацию по отдельно взятому ресурсу. Блок, передаваемый в качестве параметра, должен принимать три параметра, первый из которых должен относиться к типу `ALAsset`.
4. Получив объекты `ALAsset`, доступные в каждой группе, вы можете затем найти различные свойства каждого ресурса: его тип, доступные URL и т. д. Для получения этих свойств примените метод экземпляра `valueForProperty:`, относящийся к каждому ресурсу типа `ALAsset`. Возвращаемое значение этого метода в зависимости от типа передаваемого ему параметра может представлять собой `NSDictionary`, `NSString` или любой другой тип объекта. Вскоре мы рассмотрим общие свойства, которые могут быть получены от любого ресурса.
5. Вызовите метод экземпляра `defaultRepresentation` каждого объекта типа `ALAsset`, чтобы получить его объект представления (Representation Object) типа `ALAssetRepresentation`. Каждый ресурс из библиотеки ресурсов может иметь более одного представления. Например, фотография по умолчанию может иметь представление в формате PNG, но, кроме того, обладать и представлением JPEG. С помощью метода `defaultRepresentation` каждого ресурса типа `ALAsset` можно получить объект `ALAssetRepresentation`, а потом использовать его для получения различных представлений каждого ресурса (при наличии таких представлений).
6. Пользуйтесь методами `size` и методами экземпляра `getBytes:fromOffset:length:error:` каждого представления ресурса для загрузки данных о представлении ресурса. Затем можно записать прочитанные байты в объект `NSData` или выполнить любую другую операцию, необходимую в ходе работы приложения. Кроме того, при работе с фотографиями можно использовать методы экземпляра `fullResolutionImage`, `fullScreenImage` и `CGImageWithOptions:` каждого представления, чтобы получать изображения типа `CGImageRef`. Потом можно создать `UIImage` из `CGImageRef` с помощью метода класса `imageWithCGImage:`, относящегося к классу `UIImage`:

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

  static BOOL beenHereBefore = NO;

  if (beenHereBefore){
    /* Отображаем элемент для выбора даты только после того, как вызывается
     метод viewDidAppear:, что происходит при каждом отображении вида
     нашего контроллера вида */
    return;
  } else {
```

```
    beenHereBefore = YES;
  }
  self.assetsLibrary = [[ALAssetsLibrary alloc] init];

  [self.assetsLibrary
   enumerateGroupsWithTypes:ALAssetsGroupAll
   usingBlock:^(ALAssetsGroup *group, BOOL *stop) {
     [group enumerateAssetsUsingBlock:^(ALAsset *result,
                                        NSUInteger index,
                                        BOOL *stop) {

       /* Получаем тип ресурса. */
       NSString *assetType = [result valueForProperty:ALAssetPropertyType];

       if ([assetType isEqualToString:ALAssetTypePhoto]){
         NSLog(@"This is a photo asset");
       }

       else if ([assetType isEqualToString:ALAssetTypeVideo]){
         NSLog(@"This is a video asset");
       }

       else if ([assetType isEqualToString:ALAssetTypeUnknown]){
         NSLog(@"This is an unknown asset");
       }

       /* Получаем все URL для ресурса. */
       NSDictionary *assetURLs =
                    [result valueForProperty:ALAssetPropertyURLs];

       NSUInteger    assetCounter = 0;
       for (NSString *assetURLKey in assetURLs){
         assetCounter++;
         NSLog(@"Asset URL %lu = %@",
               (unsigned long)assetCounter,
               [assetURLs valueForKey:assetURLKey]);
       }

       /* Получаем объект представления ресурса. */
       ALAssetRepresentation *assetRepresentation =
        [result defaultRepresentation];

       NSLog(@"Representation Size = %lld", [assetRepresentation size]);

     }];
   }
   failureBlock:^(NSError *error) {
     NSLog(@"Failed to enumerate the asset groups.");
   }];

}
```

## Обсуждение

Библиотека ресурсов подразделяется на группы. В каждой группе содержатся ресурсы, а каждый ресурс имеет свойства, например URL (универсальные локаторы ресурсов) и объекты представления.

Все ресурсы всех типов можно получать из библиотеки ресурсов с помощью константы `ALAssetsGroupAll`. Она передается параметру `enumerateGroupsWithTypes` метода экземпляра `enumerateGroupsWithTypes:usingBlock:failureBlock:`, относящегося к объекту «Библиотека ресурсов». Вот список значений, которые можно передать этому параметру для перечисления различных групп ресурсов:

- `ALAssetsGroupAlbum` — группы, содержащие альбомы, которые были сохранены на устройстве iOS из iTunes;
- `ALAssetsGroupFaces` — группы, содержащие альбомы, в которых представлены фотографии лиц, сохраненные на устройстве iOS из iTunes;
- `ALAssetsGroupSavedPhotos` — группы, содержащие снимки, которые сохранены в библиотеке фотографий. На устройстве iOS эти ресурсы доступны также через приложение Photos (Фотографии);
- `ALAssetsGroupAll` — все группы, доступные в библиотеке ресурсов.

Теперь напишем простое приложение, которое будет получать данные о первом изображении, найденном в библиотеке ресурсов, создавать `UIImageView` с этим изображением, а потом добавлять это изображение в вид того контроллера вида, который отображается в настоящий момент. На данном примере мы научимся считывать содержимое ресурса, используя его представление.

Когда контроллер вида отображает свой вид, мы инициализируем объект библиотеки ресурсов и начнем перечисление ресурсов в этой библиотеке, пока не найдем первую фотографию. На данном этапе будем использовать представление этого ресурса (фотографии), чтобы отобразить фотографию в виде с изображением:

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

  static BOOL beenHereBefore = NO;

  if (beenHereBefore){
    /* Отображаем элемент для выбора даты только после того, как вызывается
     метод viewDidAppear:, что происходит при каждом отображении вида
     нашего контроллера вида */
    return;
  } else {
    beenHereBefore = YES;
  }

  self.assetsLibrary = [[ALAssetsLibrary alloc] init];

  dispatch_queue_t dispatchQueue =
    dispatch_get_global_queue(DISPATCH_QUEUE_PRIORITY_DEFAULT, 0);
```

```
dispatch_async(dispatchQueue, ^(void) {

  [self.assetsLibrary
   enumerateGroupsWithTypes:ALAssetsGroupAll
   usingBlock:^(ALAssetsGroup *group, BOOL *stop) {

     [group enumerateAssetsUsingBlock:^(ALAsset *result,
                                        NSUInteger index,
                                        BOOL *stop) {
       __block BOOL foundThePhoto = NO;
       if (foundThePhoto){
         *stop = YES;
       }

       /* Получаем тип ресурса. */
       NSString *assetType = [result
                 valueForProperty:ALAssetPropertyType];

       if ([assetType isEqualToString:ALAssetTypePhoto]){
         NSLog(@"This is a photo asset");

         foundThePhoto = YES;
         *stop = YES;

         /* Получаем объект представления ресурса. */
         ALAssetRepresentation *assetRepresentation =
          [result defaultRepresentation];

         /* Нам требуются данные о масштабе и ориентации, чтобы можно
            было создать правильно расположенное и вымеренное изображение
            UIImage из нашего объекта представления. */
         CGFloat imageScale = [assetRepresentation scale];

         UIImageOrientation imageOrientation =
          (UIImageOrientation)[assetRepresentation orientation];

         dispatch_async(dispatch_get_main_queue(), ^(void) {

           CGImageRef imageReference =
            [assetRepresentation fullResolutionImage];

           /* Сейчас создаем изображение. */
           UIImage     *image =
           [[UIImage alloc] initWithCGImage:imageReference
                                      scale:imageScale
                                orientation:imageOrientation];
           if (image != nil){
             self.imageView = [[UIImageView alloc]
                               initWithFrame:self.view.bounds];
             self.imageView.contentMode = UIViewContentModeScaleAspectFit;
             self.imageView.image = image;
```

```
            [self.view addSubview:self.imageView];

          } else {
            NSLog(@"Failed to create the image.");
          }
        });
      }
    }];
  }
  failureBlock:^(NSError *error) {
    NSLog(@"Failed to enumerate the asset groups.");
  }];
});
}
```

Мы перечисляем группы и каждый ресурс в группе. Потом находим первый ресурс-фотографию и получаем его представление. С помощью этого представления создаем `UIImage`, а уже из `UIImage` делаем `UIImageView` для демонстрации изображения в виде. Ничего сложного, правда?

Работа с видеофайлами строится немного иначе, поскольку в классе `ALAsset-Representation` нет каких-либо методов, способных возвращать объект, заключающий в себе видеофайл. Поэтому нам потребуется считать содержимое видеоресурса в буфер и, возможно, сохранить его в каталоге Documents, где впоследствии к этому документу будет проще получить доступ. Разумеется, конкретные требования зависят от определенного приложения, но в приведенном далее примере кода мы пойдем дальше — найдем первый видеофайл в библиотеке ресурсов и сохраним его в каталоге Documents в приложении под названием `Temp.MOV`:

```
- (NSString *) documentFolderPath{
  NSFileManager *fileManager = [[NSFileManager alloc] init];
  NSURL *url = [fileManager URLForDirectory:NSDocumentDirectory
                                   inDomain:NSUserDomainMask
                          appropriateForURL:nil
                                     create:NO
                                      error:nil]
    return url.path;

}

- (void)viewDidAppear:(BOOL)animated{
  [super viewDidAppear:animated];
  static BOOL beenHereBefore = NO;
  if (beenHereBefore){
    /* Отображаем элемент для выбора даты только после того, как вызывается
    метод viewDidAppear:, что происходит при каждом отображении вида
    нашего контроллера вида */
    return;
  } else {
    beenHereBefore = YES;
  }
```

```
self.assetsLibrary = [[ALAssetsLibrary alloc] init];

dispatch_queue_t dispatchQueue =
dispatch_get_global_queue(DISPATCH_QUEUE_PRIORITY_DEFAULT, 0);

dispatch_async(dispatchQueue, ^(void) {

  [self.assetsLibrary
   enumerateGroupsWithTypes:ALAssetsGroupAll
   usingBlock:^(ALAssetsGroup *group, BOOL *stop) {

     __block BOOL foundTheVideo = NO;
     [group enumerateAssetsUsingBlock:^(ALAsset *result,
                                        NSUInteger index,
                                        BOOL *stop) {

       /* Получаем тип ресурса. */
       NSString *assetType = [result
                 valueForProperty:ALAssetPropertyType];

       if ([assetType isEqualToString:ALAssetTypeVideo]){
         NSLog(@"This is a video asset");

         foundTheVideo = YES;
         *stop = YES;

         /* Получаем объект представления ресурса. */
         ALAssetRepresentation *assetRepresentation =
          [result defaultRepresentation];

         const NSUInteger BufferSize = 1024;
         uint8_t buffer[BufferSize];
         NSUInteger bytesRead = 0;
         long long currentOffset = 0;
         NSError *readingError = nil;

         /* Создаем путь, по которому должно быть сохранено видео. */
         NSString *videoPath = [documentsFolder
                             stringByAppendingPathComponent:@"Temp.MOV"];

         NSFileManager *fileManager = [[NSFileManager alloc] init];

         /* Создаем файл, если он еще не существует. */
         if ([fileManager fileExistsAtPath:videoPath] == NO){
           [fileManager createFileAtPath:videoPath
                                contents:nil
                              attributes:nil];
         }

         /* Этот дескриптор файла мы будем использовать для записи
            медийных ресурсов на диск. */
```

```
            NSFileHandle *fileHandle = [NSFileHandle
                                  fileHandleForWritingAtPath:videoPath];
            do{
              /* Считываем столько байтов, сколько можем поместить в буфер. */
              bytesRead = [assetRepresentation getBytes:(uint8_t *)&buffer
                                         fromOffset:currentOffset
                                             length:BufferSize
                                              error:&readingError];

              /* Если ничего считать не можем, то выходим из этого цикла. */
              if (bytesRead == 0){
                break;
              }

              /* Данные о смещении буфера должны быть актуальными. */
              currentOffset += bytesRead;

              /* Помещаем буфер в объект NSData. */
              NSData *readData = [[NSData alloc]
                                  initWithBytes:(const void *)buffer
                                  length:bytesRead];

              /* И записываем данные в файл. */
              [fileHandle writeData:readData];

            } while (bytesRead > 0);
            NSLog(@"Finished reading and storing the \
                  video in the documents folder");
          }
        }];
        if (foundTheVideo){
          *stop = YES;
        }
      }
      failureBlock:^(NSError *error) {
        NSLog(@"Failed to enumerate the asset groups.");
      }];
  });
}
```

Вот что происходит в данном коде.

1. Мы получаем стандартное представление первого видеоресурса, который находим в библиотеке ресурсов.
2. Создаем файл `Temp.MOV` в каталоге **Documents** нашего приложения для сохранения содержимого видеоресурса.
3. Создаем цикл, работающий до тех пор, пока в представлении ресурса еще остаются данные, которые необходимо считать. Метод экземпляра `getBytes:fromOffset:length:error:`, относящийся к объекту представления ресурса, считывает столько байтов, сколько может поместиться в буфер, и проделывает это столько раз, сколько необходимо, пока мы не достигнем конца данных представления.

4. После считывания данных в буфер мы заключаем их в объект типа `NSData`, используя для этого метод инициализации `initWithBytes:length:` класса `NSData`. Затем записываем эти данные в файл, созданный ранее, с помощью метода экземпляра `writeData:`, относящегося к классу `NSFileHandle`.

# 13.8. Редактирование видео на устройстве с операционной системой iOS

## Постановка задачи

Требуется, чтобы пользователь, просматривающий видео, мог редактировать видео в этом же приложении.

## Решение

Воспользуйтесь классом `UIVideoEditorController`. В приведенном здесь примере используем данный класс вместе с контроллером для выбора изображений. Сначала мы предлагаем пользователю выбрать снимок из его библиотеки фотографий. После того как выбор будет сделан, отображаем экземпляр контроллера видеоредактора, где пользователь может обрабатывать выбранное видео.

## Обсуждение

Класс `UIVideoEditorController`, содержащийся в iOS SDK, позволяет программисту вывести на экран перед пользователем специальный интерфейс для редактирования. Все, что требуется сделать, — предоставить URL видеоролика, который предполагается отредактировать, а потом отобразить в модальном виде контроллер для редактирования видео. Не допускайте наложения вида этого контроллера на любые другие виды и не изменяйте этот вид.

При вызове метода presentModalViewController:animated: сразу же после метода dismissModalViewControllerAnimated: контроллера вида приложение завершится с ошибкой времени исполнения. Необходимо дождаться, пока первый контроллер вида не будет убран с экрана, и только потом отображать второй контроллер вида. Можно пользоваться относящимся к контроллерам вида методом viewDidAppear:, помогающим определить, когда отобразится ваш вид. На данном этапе можно быть уверенным, что все модальные контроллеры видов уже убраны.

Итак, продолжим и объявим контроллер вида со всеми необходимыми свойствами:

```
#import "ViewController.h"
#import <MobileCoreServices/MobileCoreServices.h>
#import <AssetsLibrary/AssetsLibrary.h>

@interface ViewController ()
```

```
<UINavigationControllerDelegate,
UIVideoEditorControllerDelegate,
UIImagePickerControllerDelegate>
@property (nonatomic, strong) NSURL *videoURLToEdit;
@property (nonatomic, strong) ALAssetsLibrary *assetsLibrary;
@end

@implementation ViewController

<# Остаток вашего кода находится здесь  #>
```

Далее нужно обработать различные сообщения, получаемые от делегата видеоредактора в контроллере вида:

```
- (void)videoEditorController:(UIVideoEditorController *)editor
     didSaveEditedVideoToPath:(NSString *)editedVideoPath{
  NSLog(@"The video editor finished saving video");
  NSLog(@"The edited video path is at = %@", editedVideoPath);
  [editor dismissModalViewControllerAnimated:YES];
}

- (void)videoEditorController:(UIVideoEditorController *)editor
             didFailWithError:(NSError *)error{
  NSLog(@"Video editor error occurred = %@", error);
  [editor dismissModalViewControllerAnimated:YES];
}

- (void)videoEditorControllerDidCancel:(UIVideoEditorController *)editor{
  NSLog(@"The video editor was cancelled");
  [editor dismissModalViewControllerAnimated:YES];
}
```

Когда вид загрузится, мы должны будем отобразить для пользователя вид для выбора видео. Видео он может выбрать из библиотеки, а потом мы предоставим возможность его обрабатывать:

```
- (BOOL) cameraSupportsMedia:(NSString *)paramMediaType
             sourceType:(UIImagePickerControllerSourceType)paramSourceType{

  __block BOOL result = NO;

  if ([paramMediaType length] == 0){
    NSLog(@"Media type is empty.");
    return NO;
  }

  NSArray *availableMediaTypes =
  [UIImagePickerController
         availableMediaTypesForSourceType:paramSourceType];

  [availableMediaTypes enumerateObjectsUsingBlock:
   ^(id obj, NSUInteger idx, BOOL *stop) {
```

```
    NSString *mediaType = (NSString *)obj;
    if ([mediaType isEqualToString:paramMediaType]){
      result = YES;
      *stop= YES;
    }

  }];

  return result;

}

- (BOOL) canUserPickVideosFromPhotoLibrary{

  return [self cameraSupportsMedia:(__bridge NSString *)kUTTypeMovie
              sourceType:UIImagePickerControllerSourceTypePhotoLibrary];

}

- (BOOL) isPhotoLibraryAvailable{

  return [UIImagePickerController
          isSourceTypeAvailable:
          UIImagePickerControllerSourceTypePhotoLibrary];

}

- (void)viewDidAppear:(BOOL)animated{
  [super viewDidAppear:animated];
  static BOOL beenHereBefore = NO;
  if (beenHereBefore){
    /* Отображаем элемент для выбора даты только после того, как вызывается
     метод viewDidAppear:, что происходит при каждом отображении вида
     нашего контроллера вида */
    return;
  } else {
    beenHereBefore = YES;
    self.assetsLibrary = [[ALAssetsLibrary alloc] init];
  }
  if ([self isPhotoLibraryAvailable] &&
      [self canUserPickVideosFromPhotoLibrary]){

    UIImagePickerController *imagePicker =
      [[UIImagePickerController alloc] init];

    /* Задаем тип источника для библиотеки фотографий. */
    imagePicker.sourceType = UIImagePickerControllerSourceTypePhotoLibrary;

    /* Требуется, чтобы пользователь мог выбирать видеоролики
       из библиотеки. */
```

```
    NSArray  *mediaTypes = [[NSArray alloc] initWithObjects:
                            (__bridge NSString *)kUTTypeMovie, nil];
    imagePicker.mediaTypes = mediaTypes;

    /* Задаем делегат для текущего контроллера вида. */
    imagePicker.delegate = self;

    /* Отображаем окно для выбора изображений. */
    [self.navigationController presentModalViewController:imagePicker
                                                 animated:YES];
  }
}
```

Далее нам необходимо узнать, когда пользователь сделает выбор (то есть выберет видеоролик). Обработаем различные сообщения делегата, обслуживающего инструмент для выбора изображений:

```
- (void)    imagePickerController:(UIImagePickerController *)picker
    didFinishPickingMediaWithInfo:(NSDictionary *)info{

  NSLog(@"Picker returned successfully.");
  NSString     *mediaType = [info objectForKey:
                             UIImagePickerControllerMediaType];

  if ([mediaType isEqualToString:(NSString *)kUTTypeMovie]){
    self.videoURLToEdit =
           [info objectForKey:UIImagePickerControllerMediaURL];
  }
  [picker dismissModalViewControllerAnimated:YES];

    /* Сначала убедимся, что редактор видео способен обрабатывать видео,
       путь к которому в папке документов мы указали. */
    if ([UIVideoEditorController canEditVideoAtPath:videoPath]){

    /* Инстанцируем редактор видео. */
    UIVideoEditorController *videoEditor =
    [[UIVideoEditorController alloc] init];

    /* Становимся делегатом видеоредактора. */
    videoEditor.delegate = self;

    /* Убеждаемся, что задали путь к видеоролику. */
    videoEditor.videoPath = videoPath;

    /* А теперь отображаем видеоредактор. */
      [self.navigationController presentModalViewController:videoEditor
                                                   animated:YES];
      self.videoURLToEdit = nil;

    } else {
      NSLog(@"Cannot edit the video at this path");
```

```
    }

  }
}];
}
- (void) imagePickerControllerDidCancel:(UIImagePickerController *)picker{

  NSLog(@"Picker was cancelled");
  self.videoURLToEdit = nil;
  [picker dismissViewControllerAnimated:YES completion:nil];
}
```

В данном примере пользователь может выбрать любое видео из библиотеки фотографий. Как только он это сделает, мы отобразим контроллер видеоредактора, указав путь к видеоролику. Этот путь передает нам инструмент для выбора видео в методе делегата.

Делегат контроллера видеоредактора получает важные сообщения о состоянии этого видеоредактора. Объект делегата должен соответствовать протоколам `UIVideoEditorControllerDelegate` и `UINavigationControllerDelegate`. В данном примере мы решили, что делегатом видеоредактора должен стать контроллер вида. Как только редактирование будет закончено, объект делегата получит от контроллера видеоредактора метод делегата `videoEditorController:didSaveEditedVideoToPath:`. Путь к отредактированному видео будет передан в параметре `didSaveEditedVideoToPath`.

Прежде чем попытаться отобразить для пользователя интерфейс видеоредактора, нужно вызывать метод класса `canEditVideoAtPath:`, относящийся к классу `UIVideoEditorController`. Мы это делаем, чтобы убедиться, что тот путь, который вы пытаетесь редактировать, доступен для обработки в контроллере. Если возвращаемое значение этого метода класса равно `YES`, можно переходить к конфигурированию и отображению интерфейса редактора видео. Если нет — идем другим путем. Возможно, потребуется отобразить для пользователя предупреждение.

## См. также

Разделы 13.6 и 13.7.

# 14 Многозадачность

## 14.0. Введение

Многозадачность — это способ работы, обеспечивающий *фоновое выполнение*. Иначе говоря, приложение может работать как обычно — выполнять задачи, порождать новые потоки, слушать уведомления, реагировать на события, — но просто ничего не отображать на экране и не взаимодействовать с пользователем каким-либо образом. Когда пользователь нажимает на устройстве кнопку Home (Домой) — в более ранних версиях iPhone и iPad эта кнопка завершала работу приложения, — программа просто переходит в фоновый режим.

Когда ваше приложение переходит в фоновый режим (например, при нажатии кнопки Home (Домой)), а затем возвращается на передний план (когда пользователь вновь выбирает это приложение), система начинает посылать различные сообщения. Ожидается, что эти сообщения будут получены объектом, который мы назначаем делегатом нашего приложения. Например, когда приложение переходит в фоновый режим, делегат приложения получит метод `applicationDidEnterBackground:`. Аналогично, когда пользователь вновь вернет приложение в приоритетный режим, то есть возобновит с ним работу, делегат приложения получит делегатное сообщение `applicationWillEnterForeground:`.

Кроме этих сообщений делегатов iOS также посылает работающему приложению уведомления, когда переводит эту программу в фоновый режим или возвращает обратно в приоритетный режим. При переходе приложения в фоновый режим посылается уведомление `UIApplicationDidEnterBackgroundNotification`, а при переходе из фонового режима в приоритетный — уведомление `UIApplicationWillEnterForeground-Notification`. Для регистрации этих уведомлений можно использовать центр уведомлений, задаваемый по умолчанию.

## 14.1. Обнаружение доступности многозадачности

### Постановка задачи

Необходимо выяснить, поддерживается ли многозадачность на том устройстве с iOS, на котором работает ваше приложение.

## Решение

Вызовите метод экземпляра `isMultitaskingSupported`, относящийся к классу `UIDevice`:

```
- (BOOL) isMultitaskingSupported{

  BOOL result = NO;
  if ([[UIDevice currentDevice]
       respondsToSelector:@selector(isMultitaskingSupported)]){
    result = [[UIDevice currentDevice] isMultitaskingSupported];
  }
  return result;

}

- (BOOL)            application:(UIApplication *)application
  didFinishLaunchingWithOptions:(NSDictionary *)launchOptions{

  if ([self isMultitaskingSupported]){
    NSLog(@"Multitasking is supported.");
  } else {
    NSLog(@"Multitasking is not supported.");
  }

  self.window = [[UIWindow alloc] initWithFrame:
                 [[UIScreen mainScreen] bounds]];

  self.window.backgroundColor = [UIColor whiteColor];
  [self.window makeKeyAndVisible];
  return YES;
}
```

## Обсуждение

В зависимости от того, на работу в какой версии iOS рассчитано ваше приложение, его можно запускать и выполнять на различных устройствах, где установлены разные версии iOS. Например, вы можете разрабатывать приложение в последней версии iOS SDK, но в качестве целевой (наиболее ранняя версия iOS, в которой сможет работать ваше приложение) задать более низкую версию. В этом случае приложение сможет работать на устройстве с более ранней версией операционной системы, но, возможно, это устройство не будет поддерживать многозадачность. Золотое правило, действующее как в программировании, так и в жизни вообще (не подумайте, что я берусь философствовать), таково: если вы пытаетесь гадать, то рано или поздно ошибетесь. Поэтому не пытайтесь угадать, на каких устройствах в данное время работает ваше приложение. Ведь если iOS-разработчик указывает в Xcode минимальную поддерживаемую версию iOS, он тем самым автоматически ограничивает свою пользовательскую аудиторию. Наша цель — добиться, чтобы обладатели любых устройств с iOS, существующих в настоящее время, могли пользоваться нашим приложением.

Поэтому если вы хотите задействовать в приложении для iOS новейшие возможности многозадачности, обязательно проверяйте, доступна ли многозадачность на данном устройстве. Если многозадачность недоступна, необходимо гарантировать, что приложение корректно отреагирует на это, например выберет альтернативный режим выполнения.

# 14.2. Выполнение долгосрочной задачи в фоновом режиме

## Постановка задачи

Требуется «занять» немного времени у iOS и выполнить долгосрочную задачу, пока приложение находится в фоновом режиме.

## Решение

Воспользуйтесь методом экземпляра `beginBackgroundTaskWithExpirationHandler:`, относящимся к классу `UIApplication`. После того как задача будет решена, вызовите метод экземпляра `endBackgroundTask:`, относящийся к классу `UIApplication`.

## Обсуждение

Когда приложение переходит в фоновый режим, работа его основного потока приостанавливается. Потоки, которые вы создаете в своем приложении с помощью метода класса `detachNewThreadSelector:toTarget:withObject:`, относящегося к классу `NSThread`, также приостанавливаются. Если вы пытаетесь решить долгосрочную задачу за то время, пока приложение находится в фоновом режиме, необходимо вызвать метод экземпляра `beginBackgroundTaskWithExpirationHandler:`, относящийся к классу `UIApplication`, чтобы «занять» немного времени у системы iOS. Свойство `backgroundTimeRemaining` класса `UIApplication` содержит количество секунд, которые требуются приложению для завершения той или иной задачи. Если приложение не успеет завершить долгосрочную задачу до того, как истечет отведенный временной промежуток, iOS завершит приложение. За каждым вызовом метода `beginBackgroundTaskWithExpirationHandler:` должен следовать соответствующий вызов метода `endBackgroundTask:` (другой метод экземпляра `UIApplication`). Иными словами, если вы просите у iOS дополнительное время на завершение задачи, то должны и сообщить iOS, когда закончите эту задачу. Когда это будет сделано и окажется, что больше нет задач, которые следовало бы выполнить в фоновом режиме, ваше приложение перейдет в фоновый режим целиком и все его потоки будут приостановлены.

Когда приложение работает на переднем плане, свойство `backgroundTimeRemaining` класса `UIApplication` равно константе `DBL_MAX`, означающей максимальную величину, которая может содержаться в значении типа `double` (число с двойной точностью). В данном случае целочисленное значение, равное такому двойному вещественному

числу, обычно составляет -1. После того как у iOS запрашивается дополнительное время перед полной приостановкой приложения, в этом свойстве указывается количество секунд, требуемое программе для завершения задачи (или всех текущих задач).

Можно вызывать в приложении метод beginBackgroundTaskWithExpirationHandler: столько раз, сколько потребуется. Но важно учитывать, что, когда iOS возвращает в этом методе вашей программе метку (токен) или идентификатор задачи, вы должны вызвать метод endBackgroundTask:, и сделать это сразу же, как программа закончит выполнять задачу. Если не сделать этого, операционная система iOS может завершить приложение.

Если приложение работает в фоновом режиме, то не предполагается, что оно останется полностью функциональным и сможет обрабатывать «тяжелые» данные. Действительно, такие приложения рассчитаны лишь на завершение долгосрочных задач.

В качестве примера можно привести приложение, направившее вызов к API веб-службы и еще не получившее с сервера ответ от этого API. На время ожидания приложение будет отправлено в фоновый режим, и приложение сможет запросить дополнительное время для получения ответа с сервера. Как только ответ будет получен, приложение должно сохранить свое состояние, а потом отметить факт завершения задачи, вызвав метод экземпляра endBackgroundTask:, относящийся к классу UIApplication.

Рассмотрим пример. Начнем с того, что определим в делегате приложения свойство типа UIBackgroundTaskIdentifier. Кроме того, определим таймер типа NSTimer, который будет использоваться при ежесекундном выводе сообщений на консоль, после того как приложение уйдет в фоновый режим:

```
#import "AppDelegate.h"

@interface AppDelegate ()
@property (nonatomic, unsafe_unretained)
UIBackgroundTaskIdentifier backgroundTaskIdentifier;

@property (nonatomic, strong) NSTimer *myTimer;
@end

@implementation AppDelegate

<# Остаток вашего кода находится здесь #>
```

Теперь создадим таймер и назначим время перехода приложения в фоновый режим:

```
- (BOOL) isMultitaskingSupported{

  BOOL result = NO;
  if ([[UIDevice currentDevice]
       respondsToSelector:@selector(isMultitaskingSupported)]){
    result = [[UIDevice currentDevice] isMultitaskingSupported];
  }
```

```
  return result;

}

- (void) timerMethod:(NSTimer *)paramSender{

  NSTimeInterval backgroundTimeRemaining =
    [[UIApplication sharedApplication] backgroundTimeRemaining];

  if (backgroundTimeRemaining == DBL_MAX){
    NSLog(@"Background Time Remaining = Undetermined");
  } else {
    NSLog(@"Background Time Remaining = %.02f Seconds",
          backgroundTimeRemaining);
  }

}

- (void)applicationDidEnterBackground:(UIApplication *)application{

  if ([self isMultitaskingSupported] == NO){
    return;
  }

  self.myTimer =
  [NSTimer scheduledTimerWithTimeInterval:1.0f
                                   target:self
                                 selector:@selector(timerMethod:)
                                 userInfo:nil
                                  repeats:YES];

  self.backgroundTaskIdentifier =
  [application beginBackgroundTaskWithExpirationHandler:^(void) {
    [self endBackgroundTask];
  }];

}
```

Как видите, в обработчике завершения (Completion Handler) фоновой задачи мы вызываем метод `endBackgroundTask` делегата приложения. Этот метод написали мы сами, и выглядит он так:

```
- (void) endBackgroundTask{
  dispatch_queue_t mainQueue = dispatch_get_main_queue();

__weak AppDelegate *weakSelf = self;
  dispatch_async(mainQueue, ^(void) {
    AppDelegate
      *strongSelf = weakSelf;
    if (strongSelf != nil){
      [strongSelf.myTimer invalidate];
```

```
        [[UIApplication sharedApplication]
         endBackgroundTask:self.backgroundTaskIdentifier];
        strongSelf.backgroundTaskIdentifier = UIBackgroundTaskInvalid;
      }

  });

}
```

Есть еще пара моментов, которые нужно дополнительно уладить после завершения долгосрочной задачи:

- завершить все потоки и таймеры независимо от того, являются они таймерами из фреймворка Core Foundation или созданы с помощью GCD;
- завершить фоновую задачу, вызвав метод `endBackgroundTask:` класса `UIApplication`;
- пометить задачу как завершенную, присвоив значение `UIBackgroundTaskInvalid` идентификаторам задачи.

И последнее, но немаловажное. Если приложение выходит в приоритетный режим, а долгосрочная задача все еще не завершена, необходимо гарантированно от нее избавиться:

```
- (void)applicationWillEnterForeground:(UIApplication *)application{

  if (self.backgroundTaskIdentifier != UIBackgroundTaskInvalid){
    [self endBackgroundTask];
  }

}
```

В нашем примере, как только приложение переходит в фоновый режим, мы запрашиваем дополнительное время на завершение долгосрочной задачи (в данном случае, например, для выполнения кода нашего таймера). В то время мы регулярно считываем значение свойства `backgroundTimeRemaining` экземпляра класса `UIApplication` и выводим найденное значение на консоль. В методе экземпляра `beginBackgroundTaskWithExpirationHandler:`, который относится к классу `UIApplication`, мы записали код, который будет выполнен прямо перед тем, как закончится дополнительное время, выделенное приложению на выполнение долгосрочной задачи. (Как правило, это происходит за 5–10 секунд до истечения времени, выделенного на выполнение задачи.) Здесь мы можем просто завершить задачу, вызвав метод экземпляра `endBackgroundTask:`, относящийся к классу `UIApplication`.

Если приложение перешло в фоновый режим и запросило у операционной системы дополнительное время на исполнение кода еще до того, как отведенное время истекло, пользователь может «оживить» приложение и вернуть его в приоритетный режим. Если до этого вы приказали исполнять долгосрочную задачу в фоновом режиме и как раз для этого приложение было переведено в фоновый режим, то нужно завершить долгосрочную задачу с помощью метода экземпляра endBackgroundTask:, относящегося к классу UIApplication.

## См. также

Раздел 14.1.

# 14.3. Добавление возможностей фонового обновления в приложения

## Постановка задачи

Требуется, чтобы ваше приложение могло обновлять контент в фоновом режиме, пользуясь новыми возможностями, появившимися в последнем iOS SDK.

## Решение

Добавьте в приложение возможность фонового обновления (background fetch).

## Обсуждение

Многие приложения, ежедневно поступающие на рынок App Store, обладают возможностями соединения с теми или иными серверами. Некоторые выбирают с сервера данные для обновления, другие отсылают информацию на сервер и т. д. В течение долгого времени в iOS существовал лишь один способ обновлять контент в фоновом режиме. Требовалось «занять» у iOS некоторое количество времени (об этом мы говорили в разделе 14.2), и приложение могло потратить это время на завершение своей работы в фоновом режиме. Но такой способ работы является активным. Существует и пассивный способ решения аналогичных задач, когда приложение просто «сидит», а iOS сама выделяет приложению некоторое время на обработку данных в фоновом режиме. Итак, вам требуется просто подключить к приложению такую возможность и приказать системе iOS разбудить ваше приложение в относительно спокойный момент, когда будет удобно обработать данные в фоновом режиме. Обычно при этом происходит фоновое обновление информации.

Например, вам может потребоваться загрузить новый контент. Предположим, у нас есть приложение для работы с Twitter. Открывая это приложение, пользователь в первую очередь хочет просмотреть новые твиты. До недавнего времени это можно было реализовать лишь одним способом: открыть приложение, а потом потратить некоторое время на обновление списка твитов. Но теперь система iOS может активизировать твиттерное приложение в фоновом режиме и приказать ему обновить ленту сообщений. Таким образом, когда пользователь откроет это приложение, твиты на экране уже будут обновлены.

Чтобы задействовать в приложении возможность фонового обновления, нужно перейти на вкладку Capabilities (Возможности) в настройках вашего проекта, а в области Background Modes (Фоновые режимы) установить флажок Background fetch (Фоновое обновление) (рис. 14.1).

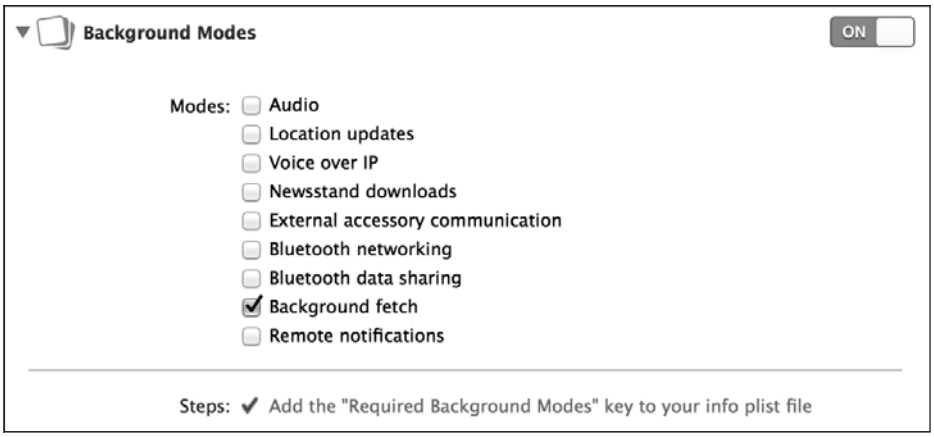


**Рис. 14.1.** Активизация фонового обновления в приложении

Приложение может использовать фоновые обновления двумя способами. Во-первых, пока приложение работает в фоновом режиме, iOS будит его и приказывает ему получить определенный контент для обновления. Во-вторых, ваше приложение может быть еще не запущено и iOS будит его (опять же в фоновом режиме) и приказывает найти контент для последующего обновления. Но как iOS узнает, какое приложение следует разбудить, а какие должны оставаться неактивизированными? В этом системе должен помочь программист.

Для этого нужно вызвать метод экземпляра `setMinimumBackgroundFetchInterval:`, относящийся к классу `UIApplication`. В качестве параметра этому методу передаются временной интервал и частота, с которой iOS должна будить ваше приложение в фоновом режиме и приказывать ему получать новые данные для обновления. По умолчанию это свойство имеет значение `UIApplicationBackgroundFetchIntervalNever`. При таком значении iOS вообще не активизирует ваше приложение в фоновом режиме. Но значение этого свойства можно установить вручную, сообщив количество секунд, образующих интервал, либо просто передать значение `UIApplicationBackgroundFetchIntervalMinimum`, при котором iOS будет «прилагать минимальные усилия» — будить ваше приложение, но делать это очень редко.

```
- (BOOL) application:(UIApplication *)application
didFinishLaunchingWithOptions:(NSDictionary *)launchOptions{

  [application setMinimumBackgroundFetchInterval:
   UIApplicationBackgroundFetchIntervalMinimum];

  return YES;
}
```

Сделав это, реализуйте метод экземпляра `application:performFetchWithCompletionHandler`, относящийся к делегату вашего приложения. Параметр `performFetchWithCompletionHandler:` этого метода дает нам блоковый объект, который нужно будет вызвать, как только приложение закончит обновлять данные. Вообще этот метод вызывается в делегате приложения, когда iOS приказывает приложению

получить в фоновом режиме новый контент для обновления. Поэтому вам придется отреагировать на это и вызвать обработчик завершения, как только все будет готово. Блоковый объект, который вам потребуется вызвать, будет принимать значение типа `UIBackgroundFetchResult`:

```
typedef NS_ENUM(NSUInteger, UIBackgroundFetchResult) {
  UIBackgroundFetchResultNewData,
  UIBackgroundFetchResultNoData,
  UIBackgroundFetchResultFailed
} NS_ENUM_AVAILABLE_IOS(7_0);
```

Итак, если iOS приказывает вашему приложению обновить контент новыми данными, вы пытаетесь получить эти данные, но их в наличии не оказывается, то вам придется вызвать обработчик завершения и передать ему значение `UIBackgroundFetchResultNoData`. Так вы сообщите iOS, что для обновления информации вашего приложения не нашлось новых данных, и система сможет откорректировать свой алгоритм планирования и механизмы искусственного интеллекта. В результате система станет вызывать приложение не столь часто. iOS действительно очень толково справляется с такими задачами. Допустим, вы приказываете iOS вызвать приложение в фоновом режиме, чтобы оно могло получить новый контент. Но сервер не дает приложению каких-либо обновлений, и в течение целой недели приложение активизируется на пользовательском устройстве в фоновом режиме, но так и не может получить новых данных и неизменно передает блоку завершения вышеупомянутого метода значение `UIBackgroundFetchResultNoData`. В таком случае совершенно очевидно, что iOS стоит будить это приложение не так часто. Так можно будет более экономно расходовать вычислительную мощность и, соответственно, заряд батареи.

В этом разделе мы собираемся написать простое приложение, которое будет получать с сервера новости. Чтобы не перегружать этот пример слишком сложным серверным кодом, мы просто сымитируем серверные вызовы. Сначала создадим класс `NewsItem`, который имеет в качестве свойств дату и текст:

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

@interface NewsItem : NSObject

@property (nonatomic, strong) NSDate *date;
@property (nonatomic, copy) NSString *text;

@end
```

У этого класса не будет никакой реализации, поэтому всю информацию он будет нести только в свойствах. Возвращаемся к делегату нашего приложения и определяем изменяемый массив новостей. Таким образом мы сможем закрепить этот массив в контроллере табличного вида и отобразить новостные элементы:

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

@interface AppDelegate : UIResponder <UIApplicationDelegate>

@property (nonatomic, strong) UIWindow *window;
```

```
@property (nonatomic, strong) NSMutableArray *allNewsItems;

@end
```

Теперь выполним отложенное выделение массива. Это означает, что массив не будет выделяться и инициализироваться до тех пор, пока приложение прямо к нему не обратится. Так экономится память. Как только массив будет выделен, мы добавим к нему один новый элемент:

```
#import "AppDelegate.h"
#import "NewsItem.h"
@implementation AppDelegate

- (NSMutableArray *) allNewsItems{
  if (_allNewsItems == nil){
    _allNewsItems = [[NSMutableArray alloc] init];

    /* Заранее записываем в массив один элемент */
    NewsItem *item = [[NewsItem alloc] init];
    item.date = [NSDate date];
    item.text = [NSString stringWithFormat:@"News text 1"];
    [_allNewsItems addObject:item];

  }
  return _allNewsItems;
}
<# Остаток кода делегата вашего приложения находится здесь #>
```

Теперь реализуем в нашем приложении метод, который будет имитировать вызов сервера. Можно сказать, что здесь мы играем в орлянку. Точнее, метод случайным образом генерирует одно из двух чисел — 0 или 1. Получив 1, мы считаем, что на сервере есть новые новостные материалы для загрузки. Получив 0, считаем, что такая новая информация на сервере отсутствует. Если мы получили 1, то сразу после этого добавляем в список новый элемент:

```
- (void) fetchNewsItems:(BOOL *)paramFetchedNewItems{

  if (arc4random_uniform(2) != 1){
    if (paramFetchedNewItems != nil){
      *paramFetchedNewItems = NO;
    }
    return;
  }

  [self willChangeValueForKey:@"allNewsItems"];

  /* Генерируем новый элемент */
  NewsItem *item = [[NewsItem alloc] init];
  item.date = [NSDate date];
  item.text = [NSString stringWithFormat:@"News text %lu",
  (unsigned long)self.allNewsItems.count + 1];
  [self.allNewsItems addObject:item];
```

```
  if (paramFetchedNewItems != nil){
    *paramFetchedNewItems = YES;
  }

  [self didChangeValueForKey:@"allNewsItems"];

}
```

Логический параметр указателя этого метода сообщит нам, появилась ли новая информация, добавленная в массив.

Теперь реализуем механизм фонового обновления в делегате нашего приложения, так, как было объяснено ранее:

```
- (void)                application:(UIApplication *)application
  performFetchWithCompletionHandler:(void (^)(UIBackgroundFetchResult))
  completionHandler{

    BOOL haveNewContent = NO;
    [self fetchNewsItems:&haveNewContent];

    if (haveNewContent){
      completionHandler(UIBackgroundFetchResultNewData);
    } else {
      completionHandler(UIBackgroundFetchResultNoData);
    }

}
```

Отлично. В контроллере нашего табличного вида отслеживаем изменения массива новостных элементов в делегате приложения. Как только содержимое массива изменится, мы обновим табличный вид. Но будем делать это с умом. Если приложение работает в фоновом режиме, то действительно следует обновить табличный вид. Но если приложение работает в фоновом режиме, отложим обновление до тех пор, пока табличный вид не перейдет в приоритетный режим:

```
#import "TableViewController.h"
#import "AppDelegate.h"
#import "NewsItem.h"

@interface TableViewController ()
@property (nonatomic, weak) NSArray *allNewsItems;
@property (nonatomic, unsafe_unretained) BOOL mustReloadView;
@end

@implementation TableViewController

- (void)viewDidLoad{
  [super viewDidLoad];

  AppDelegate *appDelegate = [UIApplication sharedApplication].delegate;
```

```
  self.allNewsItems = appDelegate.allNewsItems;

  [appDelegate addObserver:self
                forKeyPath:@"allNewsItems"
                   options:NSKeyValueObservingOptionNew
                   context:NULL];

  [[NSNotificationCenter defaultCenter]
   addObserver:self
   selector:@selector(handleAppIsBroughtToForeground:)
   name:UIApplicationWillEnterForegroundNotification
   object:nil];
  }

- (void) observeValueForKeyPath:(NSString *)keyPath
                       ofObject:(id)object
                         change:(NSDictionary *)change
                        context:(void *)context{

  if ([keyPath isEqualToString:@"allNewsItems"]){
    if ([self isBeingPresented]){
      [self.tableView reloadData];
    } else {
      self.mustReloadView = YES;
    }
  }
}

- (void) handleAppIsBroughtToForeground:(NSNotification *)paramNotification{
  if (self.mustReloadView){
    self.mustReloadView = NO;
    [self.tableView reloadData];
  }
}
```

Наконец, потребуется написать необходимые методы источника данных нашего табличного вида, позволяющие записывать новые элементы в табличный вид:

```
- (NSInteger)tableView:(UITableView *)tableView
numberOfRowsInSection:(NSInteger)section{
  return self.allNewsItems.count;
}

- (UITableViewCell *)tableView:(UITableView *)tableView
cellForRowAtIndexPath:(NSIndexPath *)indexPath{

  static NSString *CellIdentifier = @"Cell";
  UITableViewCell *cell = [tableView
                           dequeueReusableCellWithIdentifier:CellIdentifier
                           forIndexPath:indexPath];

  NewsItem *newsItem = self.allNewsItems[indexPath.row];
```

```
  cell.textLabel.text = newsItem.text;

  return cell;
}

- (void) dealloc{
  AppDelegate *appDelegate = [UIApplication sharedApplication].delegate;
  [appDelegate removeObserver:self forKeyPath:@"allNewsItems"];
  [[NSNotificationCenter defaultCenter] removeObserver:self];
}
```

В данном примере кода мы извлекаем из очереди те ячейки табличного вида, которые имеют идентификатор Cell. Метод dequeueReusableCellWithIdentifier:forIndexPath: нашего табличного вида возвращает валидные ячейки, а не nil, потому что в файле раскадровки мы уже задали этот идентификатор для прототипа ячейки в табличном виде. Во время исполнения раскадровка регистрирует для iOS эту ячейку-прототип с заданным идентификатором. Поэтому мы можем извлекать ячейки из очереди, просто опираясь на данный идентификатор, и не регистрируем ячейки заранее.

Табличные виды подробно рассмотрены в главе 4.

Теперь запустите ваше приложение и нажмите кнопку Home (Главная), чтобы перевести ваше приложение в фоновый режим. Вернитесь в Xcode и в меню Debug (Отладка) выберите Simulate Background Fetch (Имитировать обновление в фоновом режиме) (рис. 14.2). Теперь вновь откройте приложение, не завершая его, и посмотрите, появится ли в табличном виде новый контент. Если не появится — то именно по той причине, что запрограммированная нами логика напоминает игру в орлянку. Приложение случайным образом «определяет», есть ли на «сервере» новый контент. Так имитируются вызовы сервера. Если не получите никакого «нового контента», просто повторите имитацию фонового обновления в меню Debug (Отладка), пока «информация» не будет «получена».

До сих пор мы обрабатывали в системе iOS запросы на фоновое обновление, пока приложение находилось в фоновом режиме. Но что, если работа приложения полностью завершена и фонового режима в данный момент не существует? Как нам сымитировать описанную ситуацию и определить, сработает ли наша программа? Оказывается, Apple уже и об этом позаботилась. Выберите пункт Manage Schemes (Управление схемами) в меню Product (Продукт) в Xcode. Здесь скопируйте основную схему вашего приложения, нажав кнопку с плюсиком, а затем установив флажок Duplicate Scheme (Дублировать схему) (рис. 14.3).

Теперь перед вами откроется новое диалоговое окно, примерно такое, как на рис. 14.4. Здесь будет предложено установить различные свойства новой схемы. В этом диалоговом окне установите флажок Launch due to a background fetch event (Запуск, обусловленный событием фонового обновления данных), а потом нажмите OK.

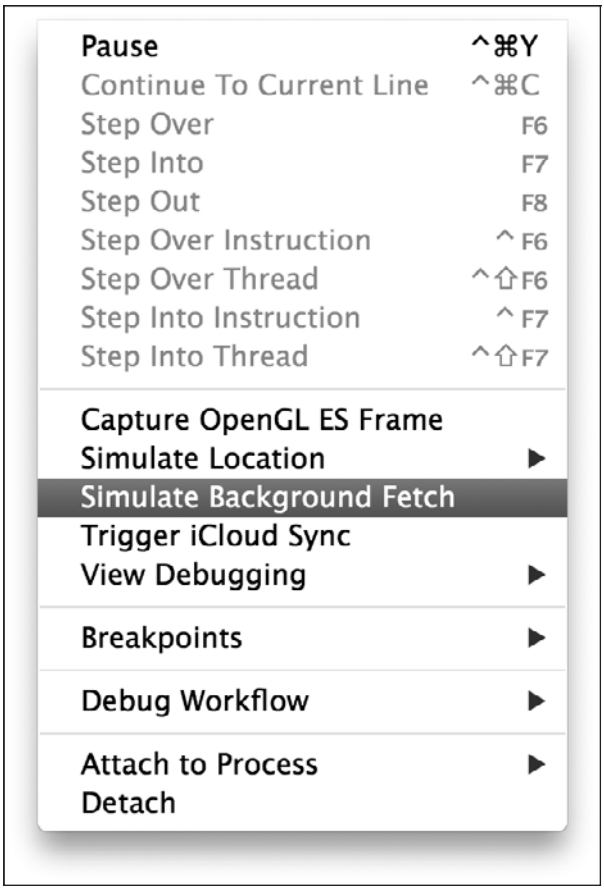


**Рис. 14.2.** Имитация фонового обновления в Xcode

Итак, теперь в Xcode записаны две схемы для приложения (рис. 14.5). Чтобы активизировать приложение с целью фонового обновления, вам просто понадобится выбрать только что созданную вторую схему и запустить приложение в симуляторе или на устройстве. В таком случае ваше приложение не перейдет в приоритетный режим. Вместо этого будет выдан сигнал для обновления данных в фоновом режиме, который, в свою очередь, вызовет метод `application:performFetchWithCompletionHandler:` делегата нашего приложения. Если вы правильно выполнили все шаги, описанные в этом разделе, то в обоих сценариях у вас будет полностью работоспособное приложение: и когда iOS вызывает программу из фонового режима, и когда приложение запускается с нуля, только для фонового обновления.

## См. также

Раздел 14.2.

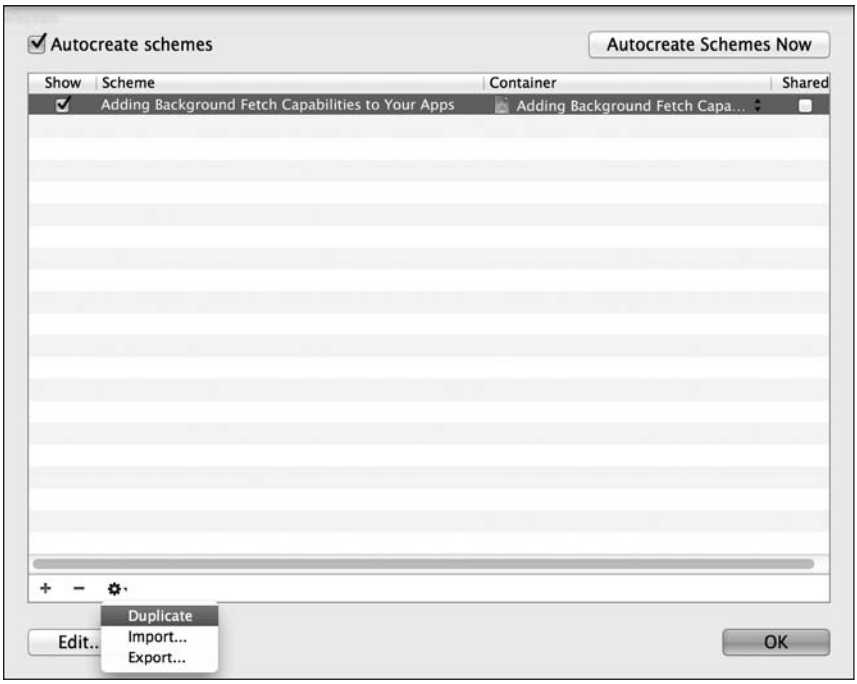


**Рис. 14.3.** Дублирование схемы для обеспечения имитации фонового обновления в период, когда приложение не работает даже в фоновом режиме

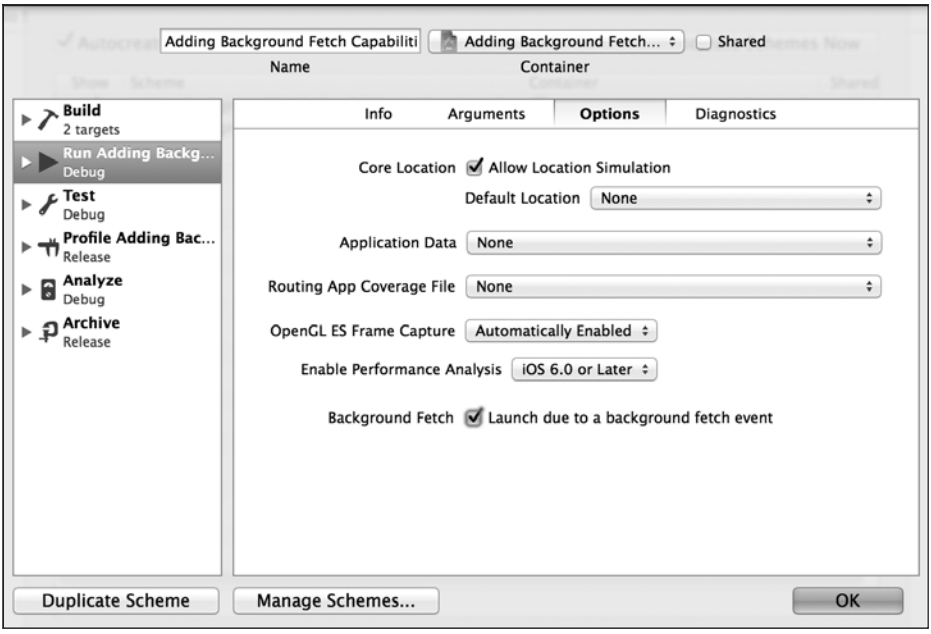


**Рис. 14.4.** Активизация схемы для запуска приложения с целью фонового обновления

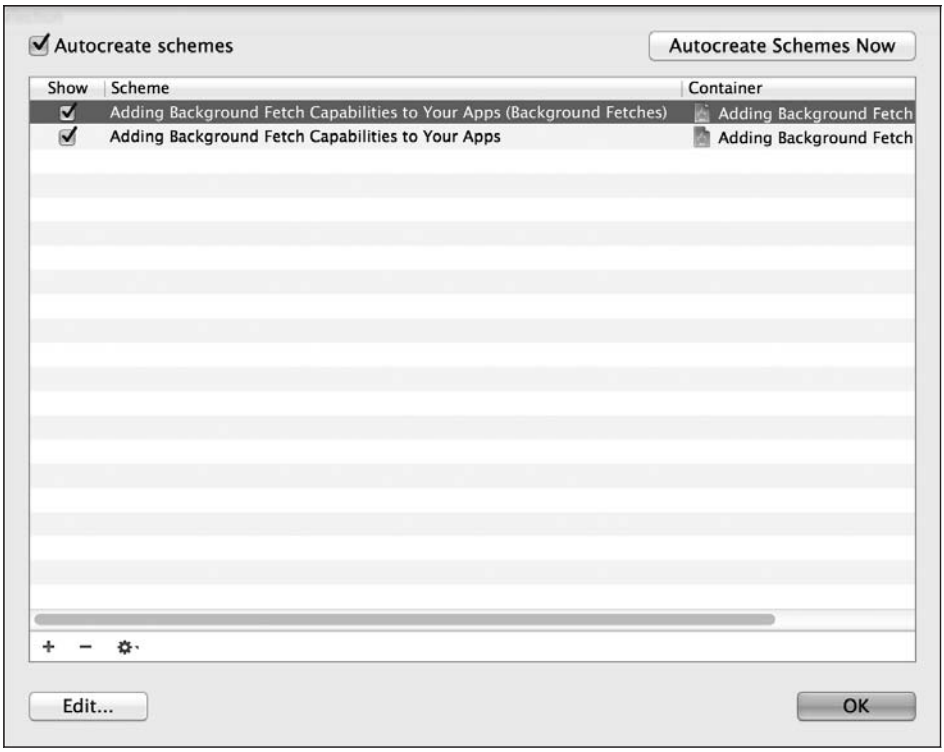


**Рис. 14.5.** Использование новой схемы для запуска вашего приложения и имитации фонового обновления данных

# 14.4. Воспроизведение аудио в фоновом режиме

## Постановка задачи

Вы пишете приложение, в котором требуется воспроизводить аудио (например, обычный музыкальный плеер), и хотите, чтобы эти файлы могли воспроизводиться даже в том случае, когда это приложение работает в фоновом режиме.

## Решение

Выберите приложение в навигаторе (Navigator) в Xcode. Затем в разделе Capabilities (Возможности) перейдите в подраздел Background Modes (Фоновые режимы). После того как система выведет список фоновых режимов, нажмите переключатель Audio (Аудио).

Теперь можно использовать фреймворк AV Foundation для воспроизведения аудиофайлов. Аудиофайлы будут проигрываться, даже если приложение работает в фоновом режиме.

Учтите, что фоновое воспроизведение аудио может не сработать в симуляторе iOS. Этот раздел нужно тестировать на реальном устройстве. Вполне возможно, что на симуляторе воспроизведение аудио прекратится, как только приложение перейдет в фоновый режим.

## Обсуждение

В iOS приложение может запросить продолжить воспроизведение своих аудиофайлов, даже если оно само переходит в фоновый режим. В этом разделе мы воспользуемся плеером `AVAudioPlayer`, который прост и удобен в обращении. Наша задача — запустить аудиоплеер и воспроизвести простой трек, а пока играет музыка — нажать кнопку Home (Домой) и перевести приложение в фоновый режим. Если мы внесем в файл `.plist` нашего приложения ключ `UIBackgroundModes`, iOS продолжит воспроизводить музыку из аудиоплеера приложения, действующего в фоновом режиме. Пока плеер работает в фоновом режиме, мы должны просто воспроизводить музыку и предоставлять плееру те данные, которые необходимы ему для работы. Нам не придется выполнять никаких иных задач, например отображать новые экраны.

Вот объявление простого делегата приложения, запускающего `AVAudioPlayer`:

```
#import "AppDelegate.h"
#import <AVFoundation/AVFoundation.h>

@interface AppDelegate () <AVAudioPlayerDelegate>
@property (nonatomic, strong) AVAudioPlayer *audioPlayer;
@end
@implementation AppDelegate

<# Остаток вашего кода находится здесь #>
```

Когда приложение откроется, мы выделим и инициализируем аудиоплеер, считаем содержимое файла `MySong.mp4` в экземпляр `NSData` и используем эти данные в процессе инициализации аудиоплеера:

```
- (BOOL)            application:(UIApplication *)application
  didFinishLaunchingWithOptions:(NSDictionary *)launchOptions{

  dispatch_queue_t dispatchQueue =
    dispatch_get_global_queue(DISPATCH_QUEUE_PRIORITY_DEFAULT, 0);

  dispatch_async(dispatchQueue, ^(void) {
    NSError *audioSessionError = nil;
    AVAudioSession *audioSession = [AVAudioSession sharedInstance];
    if ([audioSession setCategory:AVAudioSessionCategoryPlayback
                            error:&audioSessionError]){
      NSLog(@"Successfully set the audio session.");
    } else {
      NSLog(@"Could not set the audio session");
    }

    NSBundle *mainBundle = [NSBundle mainBundle];
```

```
    NSString *filePath = [mainBundle pathForResource:@"MySong"
                                              ofType:@"mp3"];

    NSData   *fileData = [NSData dataWithContentsOfFile:filePath];

    NSError  *error = nil;

    /* Запускаем аудиоплеер. */
    self.audioPlayer = [[AVAudioPlayer alloc] initWithData:fileData
                                                     error:&error];

    /* Получили ли мы экземпляр AVAudioPlayer? */
    if (self.audioPlayer != nil){
      /* Задаем делегат и начинаем воспроизведение. */

      self.audioPlayer.delegate = self;

      if ([self.audioPlayer prepareToPlay] &&
          [self.audioPlayer play]){
        NSLog(@"Successfully started playing...");

      } else {
        NSLog(@"Failed to play.");
      }

    } else {
       /* Не удалось инстанцировать AVAudioPlayer. */
    }
  });

  self.window = [[UIWindow alloc] initWithFrame:
                 [[UIScreen mainScreen] bounds]];

  self.window.backgroundColor = [UIColor whiteColor];
  [self.window makeKeyAndVisible];

  return YES;
}
```

В данном примере кода мы используем аудиосессии из фреймворка AV, чтобы сначала перевести в беззвучный режим другие приложения, воспроизводящие музыку (например, приложение iPod), и лишь потом переходим к воспроизведению аудио. Если тот аудиофайл, который воспроизводится в настоящее время (в фоновом режиме), завершается, можно запустить новый экземпляр `AVAudioPlayer` и приступить к проигрыванию нового аудиофайла. iOS откорректирует обработку информации с учетом такой ситуации. Но нет гарантии, что ваше приложение, работающее в фоновом режиме, получит от системы разрешение на выделение достаточного количества памяти, чтобы загрузить в нее данные нового аудиофайла.

Вы, наверное, заметили, что в приведенном примере кода мы делаем делегат нашего приложения делегатом аудиоплеера. Мы реализуем методы делегата аудиоплеера вот так:

```
- (void)audioPlayerBeginInterruption:(AVAudioPlayer *)player{

  /* Аудиосессия прервана.
  Здесь мы ставим плеер на паузу */

}

- (void)audioPlayerEndInterruption:(AVAudioPlayer *)player
withOptions:(NSUInteger)flags{

  /* Проверяем по флагам, можем ли мы возобновить воспроизведение аудио.
  Если да, то делаем это здесь */

  if (flags == AVAudioSessionInterruptionOptionShouldResume){
    [player play];
  }

}

- (void)audioPlayerDidFinishPlaying:(AVAudioPlayer *)player
successfully:(BOOL)flag{

  NSLog(@"Finished playing the song");

  /* Параметр flag сообщает, удалось ли успешно закончить воспроизведение */

  if ([player isEqual:self.audioPlayer]){
    self.audioPlayer = nil;
  } else {
    /* Это не наш аудиоплеер! */
  }

}
```

Кроме того, необходимо учитывать, что, когда приложение воспроизводит аудиофайл в фоновом режиме, не будет изменяться значение, возвращаемое свойством `backgroundTimeRemaining` класса `UIApplication`. Иными словами этот аспект можно описать так: приложение, запрашивающее возможность воспроизведения аудио в фоновом режиме, не запрашивает у операционной системы iOS, явно или неявно, дополнительное время на исполнение кода.

# 14.5. Обработка геолокационных изменений в фоновом режиме

## Постановка задачи

Вы пишете приложение, основной функционал которого заключается в обработке геолокационных изменений с помощью фреймворка Core Location. Необходимо,

чтобы создаваемое приложение получало данные об изменении местоположения устройства с iOS, даже если приложение переходит в фоновый режим.

## Решение

Выберите приложение в навигаторе (Navigator) в Xcode. Затем в разделе Capabilities (Возможности) перейдите в подраздел Background Modes (Фоновые режимы). После того как система выведет список фоновых режимов, нажмите переключатель Location (Местоположение).

## Обсуждение

Когда приложение работает в приоритетном режиме, можно получать от экземпляра `CLLocationManager` делегатные сообщения, информирующие вас о том, что iOS обнаружила перемещение устройства на новое место. Однако если приложение переходит в фоновый режим и становится неактивным, то делегатные сообщения не будут доставляться к нему в обычном порядке. В таком случае все делегатные сообщения поступят в программу одним пакетом, как только она снова перейдет в приоритетный режим.

Если для работы приложения вам необходима возможность получать информацию об изменении местоположения пользовательского устройства, даже если программа работает в фоновом режиме, то нужно добавить значение `location` к ключу `UIBackgroundModes` в основной файл `.plist` вашего приложения, как показано в подразделе «Решение» данного раздела. В таком случае, даже будучи в фоновом режиме, ваше приложение продолжит получать информацию об изменении местоположения устройства. Протестируем простое приложение, в котором у нас будет только делегат.

В этом приложении я собираюсь сохранить логическое значение в делегате приложения, которое будет называться `executingInBackground`. Когда приложение переходит в фоновый режим, задам этому делегату значение `YES`, а когда оно вернется в приоритетный режим — изменю данное значение на `NO`. Когда будут получены данные об изменении геолокационной информации от Core Location, мы проверим этот флаг. Если флаг установлен в `YES`, то мы не будем заниматься никакими энергоемкими вычислениями или любыми обновлениями пользовательского интерфейса, поскольку наше приложение сейчас работает в фоновом режиме. Мы, будучи ответственными программистами, не будем нагружать приложение никакими тяжелыми задачами. Но если программа действует в приоритетном режиме, то в нашем распоряжении вся вычислительная мощность устройства, которую мы можем бросить на обработку необходимых функций. Кроме того, попытаемся добиться максимальной точности при определении местоположения, если наша программа работает в приоритетном режиме. Когда же программа уходит в фоновый режим, эту точность можно смело снизить, чтобы уменьшить нагрузку на геолокационные сенсоры. Итак, определим делегат приложения:

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

```
@interface AppDelegate () <CLLocationManagerDelegate>
@property (nonatomic, strong) UIWindow *window;
@property (nonatomic, strong) CLLocationManager *myLocationManager;
@property (nonatomic, unsafe_unretained, getter=isExecutingInBackground)
  BOOL executingInBackground;

@end
@implementation AppDelegate

<# Остаток вашего кода находится здесь #>
```

Продолжим. Создадим диспетчер местоположения и запустим его сразу после запуска приложения:

```
- (BOOL)            application:(UIApplication *)application
  didFinishLaunchingWithOptions:(NSDictionary *)launchOptions{

  self.myLocationManager = [[CLLocationManager alloc] init];
  self.myLocationManager.desiredAccuracy = kCLLocationAccuracyBest;
  self.myLocationManager.delegate = self;
  [self.myLocationManager startUpdatingLocation];

  self.window = [[UIWindow alloc] initWithFrame:
                 [[UIScreen mainScreen] bounds]];
  self.window.backgroundColor = [UIColor whiteColor];
  [self.window makeKeyAndVisible];
  return YES;
}
```

Как видите, мы указали желаемую точность определения местоположения на довольно высоком уровне. Тем не менее при переходе приложения в фоновый режим лучше снизить эту точность, чтобы снизить нагрузку на iOS:

```
- (void)applicationDidEnterBackground:(UIApplication *)application{
  self.executingInBackground = YES;

  /* Снижаем точность определения местоположения и нагрузку
     на iOS, пока работаем в фоновом режиме. */
  self.myLocationManager.desiredAccuracy = kCLLocationAccuracyHundredMeters;
}
```

Когда программа вернется из фонового режима в приоритетный, точность опять можно будет поднять до высокого уровня:

```
- (void)applicationWillEnterForeground:(UIApplication *)application{
  self.executingInBackground = NO;

  /* Теперь, когда приложение вернулось в приоритетный режим, повышаем
     точность определения местоположения. */
  self.myLocationManager.desiredAccuracy = kCLLocationAccuracyBest;
}
```

Кроме того, целесообразно было бы избежать любой интенсивной обработки в такой ситуации: приложение находится в фоновом режиме, и тут диспетчер местоположения получает обновление. Поэтому необходимо обработать метод `locationManager:didUpdateToLocation:fromLocation:` делегата нашего приложения следующим образом:

```
- (void)locationManager:(CLLocationManager *)manager
    didUpdateToLocation:(CLLocation *)newLocation
           fromLocation:(CLLocation *)oldLocation{

  if ([self isExecutingInBackground]){
     /* Работаем в фоновом режиме.
        Не выполняем никакой сложной обработки. */
  } else {
     /* Работаем в приоритетном режиме. Запускаем любые вычисления,
        какие требуются */
  }

}
```

Простое правило сводится к тому, что, работая в фоновом режиме, нужно потреблять минимальный объем памяти и вычислительной мощности, который требуется для удовлетворения нужд приложения. Поэтому, снижая точность диспетчера местоположения, пока он действует в фоновом режиме, мы снижаем и вычислительную нагрузку на iOS, которая должна доставлять приложению новую геолокационную информацию.

В зависимости от того, в какой версии симулятора iOS вы тестируете свои приложения, а также от настроек сетевого соединения и многих других факторов, влияющих на этот процесс, фоновая обработка данных о местоположении может не сработать в ходе тестирования. Рекомендуется тестировать ваши приложения, а также исходный код из этого раздела на реальном устройстве.

# 14.6. Сохранение и загрузка состояния приложений iOS, использующих многозадачность

## Постановка задачи

Необходимо, чтобы при отправке приложения для iOS в фоновый режим его состояние сохранялось и восстанавливалось, как только приложение вернется в приоритетный режим.

## Решение

Комбинируйте сообщения протокола `UIApplicationDelegate`, отправляемые делегату вашего приложения, и уведомления, которые посылает система iOS. Так вы сможете сохранять состояние приложений.

## Обсуждение

Допустим, пустое приложение iOS (то есть приложение всего с одним окном, для которого еще не написан код) впервые запускается на устройстве с iOS, поддерживающем работу в многозадачном режиме. Оно запускается именно впервые, а не возвращается из фонового режима в приоритетный. В таком случае делегат приложения `UIApplicationDelegate` будет получать следующие сообщения именно в таком порядке.

1. `application:didFinishLaunchingWithOptions:`.
2. `applicationDidBecomeActive:`.

Если пользователь нажимает кнопку Home (Домой) на своем устройстве с iOS, то делегат приложения получит следующие сообщения в таком порядке.

1. `applicationWillResignActive:`.
2. `applicationDidEnterBackground:`.

Когда приложение находится в фоновом режиме, пользователь может дважды нажать кнопку Home (Домой) и выбрать нашу программу из списка фоновых приложений. (При этом не так уж важно, каким именно образом программа оказалась в фоновом режиме. Насколько мне известно, другое приложение может запустить наше посредством различных URI-схем, которые мы можем предоставить в нашей программе.) Как только программа вернется в приоритетный режим, делегат приложения получит следующие сообщения в таком порядке.

1. `applicationWillEnterForeground:`.
2. `applicationDidBecomeActive:`.

Наряду с этими сообщениями мы будем получать разнообразные уведомления от iOS, когда приложение будет переходить в фоновый режим или возвращаться в приоритетный.

Чтобы можно было сохранять и вновь загружать состояние наших приложений, нужно тщательно взвешивать, выполнение каких задач следует приостановить, переходя в фоновый режим, а возобновить — лишь когда программа вернется в приоритетный режим. Рассмотрим пример. Сама система может с легкостью восстанавливать сетевые соединения. Поэтому мы можем ничего специально не предпринимать, если качаем файл из Сети. Но когда мы, например, пишем игру, лучше слушать те уведомления, которые iOS направляет нашей программе, работающей в фоновом режиме, и адекватно на них реагировать. В таком случае можно просто «поставить на паузу» игровой движок. Это же при необходимости можно проделать и со звуковым движком.

После того как приложение отправлено в фоновый режим, у него есть около 10 секунд, чтобы сохранить все несохраненные данные и приготовиться к тому, чтобы вернуться в приоритетный режим в любой момент, когда этого потребует пользователь. При необходимости можно также запросить у операционной системы дополнительное время на исполнение этих функций (подробнее об этом рассказано в разделе 14.2).

Рассмотрим сохранение состояния на примере. Предположим, мы пишем игру для iOS. Когда игра отправляется в фоновый режим, нам нужно сделать следующее.

1. Приостановить игровой движок.
2. Сохранить на диск очки, заработанные пользователем.
3. Сохранить на диск информацию об уровне, на котором остановилась игра. В частности, будем сохранять точку, которой пользователь достиг на этом уровне, физические аспекты уровня, положение камеры и т. д.

Когда пользователь снова откроет игру, переводя приложение в приоритетный режим, нам нужно выполнить следующее.

1. Загрузить с диска заработанные пользователем очки.
2. Загрузить с диска тот уровень, на котором пользователь прервал игру.
3. Возобновить работу игрового движка.

А теперь предположим, что делегат нашего приложения — это игровой движок. Определим в заголовочном файле делегата приложения несколько методов:

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

@interface AppDelegate : UIResponder <UIApplicationDelegate>

@property (nonatomic, strong) UIWindow *window;

/* Сохраняем состояние нашего приложения. */
- (void) saveUserScore;
- (void) saveLevelToDisk;
- (void) pauseGameEngine;

/* Загружаем состояние нашего приложения. */
- (void) loadUserScore;
- (void) loadLevelFromDisk;
- (void) resumeGameEngine;

@end
```

Переходим к работе с заглушками методов, уже присутствующими в файле реализации делегата приложения:

```
#import "AppDelegate.h"

@implementation AppDelegate

- (void) saveUserScore{
  /* Здесь сохраняем очки, заработанные пользователем. */
}

- (void) saveLevelToDisk{
  /* Сохраняем на диске текущий уровень и положение игрока
     на карте этого уровня. */
}

- (void) pauseGameEngine{
```

```
  /* Здесь приостанавливаем работу игрового движка. */
}

- (void) loadUserScore{
  /* Загружаем обратно в память местонахождение игрока. */
}

- (void) loadLevelFromDisk{
  /* Загружаем последнее местонахождение игрока на карте. */
}

- (void) resumeGameEngine{
  /* Здесь возобновляем работу игрового движка. */
}
<# Остаток вашего кода находится здесь #>
```

Теперь нужно удостовериться, что наше приложение способно обрабатывать прерывания, в частности входящие звонки, поступающие на iPhone. В таких случаях приложение не будет переходить в фоновый режим, но тем не менее станет неактивным. Если, например, пользователь закончит телефонный разговор, то iOS вернет наше приложение в активное состояние. Итак, когда приложение становится неактивным, нужно убедиться, что приостановлена работа игрового движка. Когда приложение снова активизируется, работу игрового движка можно возобновить. На самом деле, когда приложение становится неактивным, перед нами не стоит необходимость сохранять все на диске (как минимум в этом примере), так как iOS вернет приложение в предыдущее состояние лишь после того, как приложение вновь станет активным:

```
- (void)applicationWillResignActive:(UIApplication *)application{
  [self pauseGameEngine];
}

- (void)applicationDidBecomeActive:(UIApplication *)application{
  [self resumeGameEngine];
}
```

Теперь все просто. Как только наше приложение уйдет в фоновый режим, мы сохраним состояние этой программы, а когда она вернется в приоритетный режим — вновь загрузим это состояние:

```
- (void)applicationDidEnterBackground:(UIApplication *)application{
  [self saveUserScore];
  [self saveLevelToDisk];
  [self pauseGameEngine];
}

- (void)applicationWillEnterForeground:(UIApplication *)application{
  [self loadUserScore];
  [self loadLevelFromDisk];
  [self resumeGameEngine];
}
```

Разумеется, не всякое приложение — это игра. Но описанными приемами можно пользоваться для загрузки и сохранения состояния приложений в многозадачной среде iOS.

## См. также

Раздел 14.2.

# 14.7. Управление сетевыми соединениями в фоновом режиме

## Постановка задачи

Вы применяете экземпляры класса `NSURLConnection` для получения данных с веб-сервера и отправки информации на сервер. Возникает вопрос: как гарантировать работу ваших приложений в многозадачной среде iOS, надежно застраховавшись от сбоев в соединениях.

## Решение

Следует обеспечить обработку ошибок соединения в блоковых объектах, передаваемых вашим объектам соединений.

## Обсуждение

При работе с приложениями, которые используют класс `NSURLConnection`, но, уходя в фоновый режим, не запрашивают у iOS дополнительного времени, обращаться с соединениями не составляет никакого труда. Рассмотрим на примере, как будет действовать асинхронное соединение, если приложение сначала уходит в фоновый режим, а потом возвращается в приоритетный. Итак, сделаем запрос на асинхронное соединение, чтобы получить контент, расположенный по определенному URL (например, на домашней странице Apple):

```
- (BOOL)            application:(UIApplication *)application
  didFinishLaunchingWithOptions:(NSDictionary *)launchOptions{

  NSString *urlAsString = @"http://www.apple.com";
  NSURL *url = [NSURL URLWithString:urlAsString];
  NSURLRequest *urlRequest = [NSURLRequest requestWithURL:url];
  NSOperationQueue *queue = [[NSOperationQueue alloc] init];

  [NSURLConnection
   sendAsynchronousRequest:urlRequest
   queue:queue
   completionHandler:^(NSURLResponse *response, NSData *data, NSError
```

```
                    *error) {

    if ([data length] > 0 &&
        error != nil){
    /* Данные вернулись. */
    }
    else if ([data length] == 0 &&
             error != nil){
       /* Никаких данных от сервера не пришло. */
    }
    else if (error != nil){
       /* Произошла ошибка. Ее обязательно нужно правильно обработать. */
    }

  }];

  self.window = [[UIWindow alloc] initWithFrame:
                 [[UIScreen mainScreen] bounds]];

  self.window.backgroundColor = [UIColor whiteColor];
  [self.window makeKeyAndVisible];
  return YES;
}
```

В этом примере целесообразно заменить URL домашней страницы Apple другим интернет-адресом, по которому расположен какой-нибудь крупный файл. Причина заключается в том, что, пока ваше приложение будет скачивать большой файл, у вас будет больше времени поэкспериментировать с приложением — отправить его в фоновый режим, а потом вернуть в приоритетный. Если же у вас довольно быстрое соединение с Интернетом, а вы загружаете всего одну страницу Apple, то вполне вероятно, что на это уйдет всего 1–2 секунды.

Будучи в приоритетном режиме, наше приложение продолжит загрузку файла. В ходе загрузки пользователь может нажать кнопку Home (Домой) и отправить приложение в фоновый режим. И тогда вы увидите настоящее волшебство! iOS автоматически приостановит процесс загрузки без всякого вашего вмешательства. Когда же пользователь вновь переведет программу в приоритетный режим, загрузка возобновится и вам не придется писать ни единой строки кода для обработки многозадачности в такой ситуации.

Теперь рассмотрим, что происходит при синхронных соединениях. Как только наше приложение запустится, попробуем скачать очень большой файл через главный поток (крайне порочная практика, никогда так не делайте в боевом проекте!):

```
- (BOOL)            application:(UIApplication *)application
  didFinishLaunchingWithOptions:(NSDictionary *)launchOptions{

  /* Заменяем этот URL ссылкой на крупный файл. */
  NSString *urlAsString = @"http://www.apple.com";
```

```
  NSURL *url = [NSURL URLWithString:urlAsString];
  NSURLRequest *urlRequest = [NSURLRequest requestWithURL:url];
  NSError *error = nil;

  NSData *connectionData =
                       [NSURLConnection  sendSynchronousRequest:urlRequest
                                              returningResponse:nil
                                                          error:&error];

  if ([connectionData length] > 0 &&
      error == nil){

   }
  else if ([connectionData length] == 0 &&
           error == nil){

  }
  else if (error != nil){

  }

  self.window = [[UIWindow alloc] initWithFrame:
                 [[UIScreen mainScreen] bounds]];

  self.window.backgroundColor = [UIColor whiteColor];
  [self.window makeKeyAndVisible];
  return YES;
}
```

Если вы запустите это приложение и переведете его в фоновый режим, то заметите, что на задний план отходит только графический пользовательский интерфейс, но вот ядро приложения никуда из приоритетного режима не уходит и нужные сообщения делегата — `applicationWillResignActive:` и `applicationDidEnterBackground:` — так и не будут получены. Я проводил такой опыт на iPhone.

Проблема такого решения заключается в том, что для синхронной загрузки файлов мы потребляем ту долю компьютерного времени, которая отводится главному потоку. Чтобы избавиться от проблемы, мы можем либо асинхронно загружать файлы в главном потоке, как было продемонстрировано ранее, либо синхронно загружать их в отдельных потоках.

Вернемся к предыдущему примеру кода. Если мы будем загружать тот же большой файл синхронно, в глобальной параллельной очереди, то с уходом приложения в фоновый режим соединение будет приостановлено и возобновится только после возвращения программы в приоритетный режим:

```
- (BOOL)            application:(UIApplication *)application
  didFinishLaunchingWithOptions:(NSDictionary *)launchOptions{

  dispatch_queue_t dispatchQueue =
    dispatch_get_global_queue(DISPATCH_QUEUE_PRIORITY_DEFAULT, 0);
```

```
  dispatch_async(dispatchQueue, ^(void) {

    /* Заменяем этот URL ссылкой на крупный файл. */
    NSString *urlAsString = @"http://www.apple.com";
    NSURL *url = [NSURL URLWithString:urlAsString];
    NSURLRequest *urlRequest = [NSURLRequest requestWithURL:url];
    NSError *error = nil;

    NSData *connectionData = [NSURLConnection
                                        sendSynchronousRequest:urlRequest
                                             returningResponse:nil
                                                         error:&error];
    if ([connectionData length] > 0 &&
        error == nil){

    }
    else if ([connectionData length] == 0 &&
             error == nil){

    }
    else if (error != nil){

    }
  });

  self.window = [[UIWindow alloc] initWithFrame:
                 [[UIScreen mainScreen] bounds]];

  self.window.backgroundColor = [UIColor whiteColor];
  [self.window makeKeyAndVisible];
  return YES;
}
```

## См. также

Раздел 14.2.

# 14.8. Отказ от многозадачности

## Постановка задачи

Требуется исключить использование в вашем приложении многозадачности.

## Решение

Добавьте в главный файл `.plist` приложения ключ `UIApplicationExitsOnSuspend` и задайте ему значение `true`:

```
<# Некоторые ключи и значения #>
<key>UIApplicationExitsOnSuspend</key>
<true/>
<# Остальные ключи и значения #>
```

## Обсуждение

Иногда бывает необходимо исключить возможность многозадачности в приложениях для iOS. (Хотя я настоятельно рекомендую разрабатывать программы с поддержкой многозадачности.) В таких случаях нужно добавить ключ `UIApplicationExitsOnSuspend` в главный файл `.plist` приложения. Устройства с самыми новыми версиями системы iOS понимают это значение, и операционная система будет завершать приложения, не переводя их в фоновый режим, если в файле `.plist` того или иного приложения этот ключ будет иметь значение `true`. В более ранних версиях iOS, где не поддерживается многозадачность, операционная система будет просто игнорировать это значение.

Когда подобное приложение работает в новой версии iOS, оно получит следующие сообщения делегата.

1. `application:didFinishLaunchingWithOptions:`.
2. `applicationDidBecomeActive:`.

Если пользователь нажмет на устройстве кнопку Home (Домой), то делегату будут отправлены следующие сообщения.

1. `applicationDidEnterBackground:`.
2. `applicationWillTerminate:`.

# 15 Уведомления

## 15.0. Введение

Уведомления — это объекты, несущие определенную информацию, которая может передаваться множеству получателей методом широковещания. Уведомления очень удобны для разделения работы на относительно самостоятельные фрагменты кода, но при злоупотреблении ими ситуация легко может выйти из-под контроля. Следует понимать границы возможностей при работе с уведомлениями. В этой главе мы подробно поговорим об использовании уведомлений и узнаем, когда лучше обходиться без них.

В iOS доступны уведомления трех типов.

- *Обычное уведомление* (экземпляр класса `NSNotification`). Это обычное уведомление. Программа может широковещательно передавать его любым получателям в рамках приложения. iOS также широковещательно направляет вашему приложению уведомления такого типа, пока приложение находится в приоритетном режиме. Таким образом приложение получает информацию о различных системных событиях, происходящих во время его работы, например о выводе виртуальной клавиатуры на экран и ее уходе с экрана. Эти уведомления хорошо подходят для ослабления связанности кода и позволяют аккуратно отделять друг от друга различные компоненты сложного iOS-приложения.
- *Локальное уведомление* (экземпляр класса `UILocalNotification`). Это уведомление, которое должно быть доставлено вашему приложению в определенный момент времени. Приложение сможет его получить, даже если находится в фоновом режиме или не работает вообще. Если приложение не работало, но получило такое уведомление, то оно запускается. Как правило, вы назначаете локальное уведомление, если хотите гарантированно разбудить приложение (предполагается, что пользователь разрешил вам такое действие, подробнее об этом — в дальнейшем) в определенный момент дня.
- *Пуш-уведомление*. Такое уведомление отсылается на устройство iOS с сервера. Это уведомление выполняется по инициативе сервера, поэтому приложению не приходится опрашивать сервер на наличие таких уведомлений. iOS поддерживает постоянное соединение с серверами APNS (службы Apple для обеспечения

пуш-уведомлений). Как только появляется новое пуш-уведомление, iOS обрабатывает сообщение и отсылает его тому приложению, которому это уведомление предназначалось.

Далее мы будем называть обычные уведомления просто уведомлениями. Слово «обычный» в данном контексте избыточно.

Особенность локальных уведомлений заключается в том, что они видны пользователю и пользователь может совершать над ними те или иные действия. iOS зафиксирует действие пользователя, после чего прикажет вашему приложению обработать это действие. В то же время уведомления являются невидимыми элементами. В приложении их можно распространять широковещательным способом, и приложение обязано обрабатывать эти уведомления. Пользователь не обязательно должен быть непосредственно вовлечен в этот процесс, если вы сами не потребуете от него каких-либо действий в результате получения и обработки уведомления. Например, приложение может послать уведомление в другую часть этого же уведомления. По получении такого уведомления в другой части вашего приложения генерируется диалоговое окно с предупреждением. После этого уже требуется вмешательство пользователя: он должен ознакомиться с этим окном и, например, нажать в нем кнопку OK, чтобы закрыть его. Такое непрямое вовлечение пользователя очень отличается от его непосредственного участия, которое требуется при работе с обычными уведомлениями.

Уведомления — важнейшая составляющая операционных систем iOS и OS X. iOS выдает уведомления, действующие в масштабах всей системы. Эти уведомления адресуются всем приложениям в системе, которые их слушают, и сами приложения также могут отсылать уведомления. Такое уведомление, действующее в масштабах всей системы (также называемое *распределенным*), может выдаваться только самой системой iOS.

Уведомление — это простая сущность, представленная в iOS SDK классом `NSNotification`. Уведомление отсылается объектом и может нести информацию. Объект, отсылающий уведомление, «представляет себя» центру уведомления в момент самой отправки уведомления. Затем получатель уведомления может «справиться» об отправителе по его имени класса для получения более подробной информации об этом объекте. Объект-отправитель называется объектом уведомления. Кроме того, уведомление может включать в себя словарь с пользовательской информацией. Это словарная структура данных, которая может нести дополнительную информацию об уведомлении. Если словарь не предоставляется, то этот параметр равен `nil`.

# 15.1. Отправка уведомлений

## Постановка задачи

Требуется разграничить части вашего приложения и отправить уведомление, которое может быть подхвачено другим компонентом приложения.

## Решение

Создайте экземпляр класса NSNotification и широковещательно передайте его вашему приложению, воспользовавшись методом класса postNotification:. Вы можете получить экземпляр центра уведомлений, воспользовавшись его методом класса defaultCenter, вот так:

```
#import "AppDelegate.h"

NSString *const kNotificationName = @"NotificationNameGoesHere";

@implementation AppDelegate

- (BOOL) application:(UIApplication *)application
didFinishLaunchingWithOptions:(NSDictionary *)launchOptions{

  NSNotification *notification = [NSNotification
                                  notificationWithName:kNotificationName
                                  object:self
                                  userInfo:@{@"Key 1" : @"Value 1",
                                  @"Key 2" : @2}];

  [[NSNotificationCenter defaultCenter] postNotification:notification];

  self.window = [[UIWindow alloc]
                 initWithFrame:[[UIScreen mainScreen] bounds]];
  self.window.backgroundColor = [UIColor whiteColor];
  [self.window makeKeyAndVisible];
  return YES;
}
```

## Обсуждение

Объект уведомления инкапсулируется в экземпляр класса NSNotification. Сам по себе объект уведомления практически ничего не представляет. Чтобы он был полезен, его нужно послать приложению с помощью центра уведомлений. Объект уведомления имеет три важных свойства.

- *Имя*. Это строка. Когда получатель начинает слушать уведомления, он должен указать имя интересующего его уведомления, как будет показано далее в этой главе. Если вы отправляете уведомление в созданный вами класс, убедитесь, что имя уведомления хорошо документировано. Еще лучше импортировать символ этой строки в файл заголовка. Подобный пример мы рассмотрим чуть позже в этом разделе.
- *Объект-отправитель*. По желанию вы можете указать объект, являющийся отправителем уведомления. Обычно в таком качестве задается self. Но зачем же нужно указывать отправитель уведомления? Эта информация полезна для тех компонентов приложения, которые слушают уведомления. Допустим, в одном из ваших классов вы отправляете уведомление с именем MyNotification, а другой

класс приложения отсылает уведомление с точно таким же именем. Когда элемент начинает слушать уведомление с именем `MyNotification`, получатель может указать, из какого источника ожидается интересующее его уведомление. Так, получатель может указать, что ему требуются все уведомления с именем `MyNotification`, поступающие от конкретного объекта, но не интересуют одноименные уведомления, приходящие от другого объекта. Таким образом, получатель действительно контролирует ситуацию. Хотя вы даже и можете при отправке уведомления указать вместо объекта-получателя `nil`, гораздо целесообразнее задавать данному свойству `self`, то есть имя объекта, отправляющего уведомление.

- *Словарь с пользовательской информацией*. Это словарный объект, который вы можете прикреплять к объекту уведомления. Затем получатель может считывать этот словарь, когда получает уведомление. Можно сказать, что в этом параметре удобно передавать получателям вашего уведомления дополнительную информацию.

## См. также

Раздел 15.0.

# 15.2. Слушание уведомлений и реагирование на них

## Постановка задачи

Требуется отреагировать на уведомление, посылаемое либо вашим приложением, либо системой.

## Решение

Слушайте интересующее вас уведомление путем вызова метода `addObserver:selector:name:object:` стандартного центра уведомлений. Этот метод имеет следующие параметры:

- `addObserver` — объект, который должен отслеживать заданное уведомление. Поэтому, если речь идет о текущем классе, задайте здесь `self`, чтобы указать на актуальный экземпляр вашего класса;
- `selector` — селектор, который будет получать уведомление. Этот селектор должен иметь один параметр типа `NSNotification`;
- `name` — имя уведомления, которое вы хотите слушать;
- `object` — объект, который должен прислать вам уведомление. Например, если одноименные уведомления поступают сразу от двух объектов, то вы можете сузить круг интересующих вас уведомлений и слушать только те из них, которые приходят от объекта A, игнорируя при этом приходящие от объекта B.

Если вы больше не хотите получать уведомления, выполните метод экземпляра `removeObserver:`, относящийся к классу `NSNotificationCenter`. Это должно делаться лишь при условии, что центр уведомлений удерживает экземпляры объектов-слушателей. Если центр уведомлений продолжает удерживать экземпляр вашего класса после того, как он был высвобожден, могут возникать утечки памяти и ошибки. Поэтому убедитесь в том, что своевременно удаляете объект из списка наблюдателей.

## Обсуждение

Вся эта теория станет значительно более понятной, если пояснить на примере. Мы собираемся создать класс `Person` и добавить к нему два свойства: имя и фамилию. Оба этих свойства будут относиться к типу `NSString`. Затем в делегате нашего приложения мы собираемся инстанцировать объект типа `Person`. Но не будем задавать имя и фамилию этой персоны, а отошлем в центр уведомлений само уведомление и его пользовательский словарь. В этом пользовательском словаре уведомления запишем имя и фамилию как элементы типа `NSString`. В методе инициализации класса `Person` мы собираемся слушать уведомление, которое приходит от делегата приложения. Затем извлечем имя и фамилию из пользовательского словаря и зададим эти значения для соответствующих свойств объекта-персоны.

Вот заголовочный файл делегата нашего приложения:

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

/* Имя уведомления, которое мы собираемся послать */
extern NSString *const kSetPersonInfoNotification;
/* Ключ имени в словаре пользовательской информации уведомления */
extern NSString *const kSetPersonInfoKeyFirstName;
/* Ключ фамилии в словаре пользовательской информации уведомления */
extern NSString *const kSetPersonInfoKeyLastName;

@interface AppDelegate : UIResponder <UIApplicationDelegate>

@property (nonatomic, strong) UIWindow *window;

@end
```

А вот реализация делегата нашего приложения:

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

NSString *const kSetPersonInfoNotification = @"SetPersonInfoNotification";
NSString *const kSetPersonInfoKeyFirstName = @"firstName";
NSString *const kSetPersonInfoKeyLastName = @"lastName";

@implementation AppDelegate

- (BOOL) application:(UIApplication *)application
```

```
didFinishLaunchingWithOptions:(NSDictionary *)launchOptions{

  Person *steveJobs = [[Person alloc] init];

  NSNotification *notification =
  [NSNotification
   notificationWithName:kSetPersonInfoNotification
   object:self
   userInfo:@{kSetPersonInfoKeyFirstName : @"Steve",
   kSetPersonInfoKeyLastName : @"Jobs"}];

  /* В настоящее время класс person слушает это уведомление. Этот класс
   извлечет из уведомления информацию об имени и фамилии и задаст
   собственные имя и фамилию, основываясь на информации, полученной
   из пользовательского словаря уведомления. */
  [[NSNotificationCenter defaultCenter] postNotification:notification];

  /* Вот доказательство */
  NSLog(@"Person's first name = %@", steveJobs.firstName);
  NSLog(@"Person's last name = %@", steveJobs.lastName);

  self.window = [[UIWindow alloc]
                 initWithFrame:[[UIScreen mainScreen] bounds]];
  self.window.backgroundColor = [UIColor whiteColor];
  [self.window makeKeyAndVisible];
  return YES;
}
```

Важнее всего в данном случае будет реализация класса `Person` (`Person.m`):

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

@implementation Person

- (void) handleSetPersonInfoNotification:(NSNotification *)paramNotification{

  self.firstName = paramNotification.userInfo[kSetPersonInfoKeyFirstName];
  self.lastName = paramNotification.userInfo[kSetPersonInfoKeyLastName];

}

- (instancetype) init{
  self = [super init];
  if (self != nil){

    NSNotificationCenter *center = [NSNotificationCenter defaultCenter];

    [center addObserver:self
               selector:@selector(handleSetPersonInfoNotification:)
                   name:kSetPersonInfoNotification
```

```
                object:[[UIApplication sharedApplication] delegate]];

  }
  return self;
}

- (void) dealloc{
  [[NSNotificationCenter defaultCenter] removeObserver:self];
}
@end
```

Значение, указываемое для параметра object метода addObs erver:selector:name:object:, — это объект, от которого, как предполагается, должно поступать уведомление. Если какой-то другой объект пошлет уведомление с таким же именем, то ваш слушатель не должен будет обрабатывать это уведомление. Как правило, вы указываете такой объект в тех случаях, когда точно знаете, какой именно объект должен посылать интересующее вас уведомление. Такая возможность есть не всегда. Например, бывают очень сложные приложения, где один контроллер вида, расположенный на конкретной вкладке, должен слушать уведомления, поступающие от другого контроллера вида, находящегося на другой вкладке. При этом у слушателя может и не быть ссылки на тот экземпляр контроллера вида, откуда будет исходить уведомление. В таком случае можно передать nil параметру parameter вышеупомянутого метода.

Запустив это приложение, вы увидите на консоли следующий вывод:

```
Person's first name = Steve
Person's last name = Jobs
```

Итак, и отправка, и получение этого уведомления произошли внутри нашей программы. А что насчет системных уведомлений? Мы подробнее поговорим о них в дальнейшем. Сейчас, пока вы находитесь в Xcode, нажмите комбинацию клавиш Command+Shift+O (O — это Open (Открыть)), после чего введите `UIWindow.h`. Открыв этот файл заголовка, найдите в нем объект `UIKeyboardWillShowNotification`, а в этом объекте — такой блок кода:

```
// Каждое уведомление включает в себя объект nil и словарь userInfo,
// содержащий (в координатах экрана) начало и конец области, которую
// занимает на дисплее виртуальная клавиатура.
// Пользуйтесь различными возможностями convertRect, относящимися к UIView и UIWindow,
чтобы получить контур клавиатуры в желаемой системе
// координат. Анимационные пары "ключ/значение" доступны лишь для
// уведомлений, относящихся к семейству "will"
UIKIT_EXTERN NSString *const UIKeyboardWillShowNotification;
UIKIT_EXTERN NSString *const UIKeyboardDidShowNotification;
UIKIT_EXTERN NSString *const UIKeyboardWillHideNotification;
UIKIT_EXTERN NSString *const UIKeyboardDidHideNotification;
```

Это код Apple. Мы написали наш код точно таким же образом. Apple предоставляет уведомления, отправляемые системой, а затем документирует их. Вам понадобится сделать нечто подобное. При создании уведомлений, посылаемых компонентами, находящимися внутри вашего приложения, обязательно документируйте

эти уведомления и объясняйте другим программистам (в частности, коллегам, работающим с вами в одной команде), каковы типичные значения, содержащиеся в пользовательском словаре такого уведомления, а также сообщайте им всю прочую важную информацию о ваших уведомлениях.

# 15.3. Слушание уведомлений, поступающих с клавиатуры, и реагирование на них

## Постановка задачи

Мы позволяем пользователю вводить какой-либо текст в нашем графическом интерфейсе. Для этого применяется определенный компонент, например текстовое поле или текстовый вид, требующий наличия клавиатуры. Тем не менее если всплывающая на экране виртуальная клавиатура заслоняет половину пользовательского интерфейса, то она практически бесполезна. Подобной ситуации необходимо избегать.

## Решение

Нужно слушать уведомления, поступающие от клавиатуры, и перемещать компоненты пользовательского интерфейса вверх или вниз либо полностью перегруппировывать их так, чтобы пользователь мог видеть нужную ему часть графического интерфейса, даже если клавиатура и занимает половину экрана. Сами уведомления, посылаемые клавиатурой, подробнее рассматриваются в подразделе «Обсуждение» данного раздела.

## Обсуждение

У устройств, работающих с операционной системой iOS, нет физической клавиатуры. Но у них есть виртуальная клавиатура, всплывающая на экране всякий раз, когда пользователь должен ввести текст в какое-нибудь текстовое поле (`UITextField`, см. раздел 1.19) или текстовый вид (`UITextView`, см. раздел 1.20). На iPad пользователь даже может делить клавиатуру на части и перемещать их вверх и вниз. Есть несколько пограничных случаев, о которых, вам, возможно, придется позаботиться при проектировании пользовательского интерфейса. Можете обратиться к помощи дизайнеров пользовательских интерфейсов (если в вашей компании есть такие специалисты) и рассказать им, что на iPad пользователь может делить клавиатуру на части. Перед тем как приступать к творческой работе, они должны об этом узнать. В этом разделе мы коснемся подобного пограничного случая.

Сначала рассмотрим, как выглядит клавиатура в iPhone. Виртуальная клавиатура может отображаться в книжной или альбомной ориентации. В книжной ориентации клавиатура iPhone выглядит так, как на рис. 15.1.

А клавиатура в альбомной ориентации на iPhone будет выглядеть так, как на рис. 15.2.

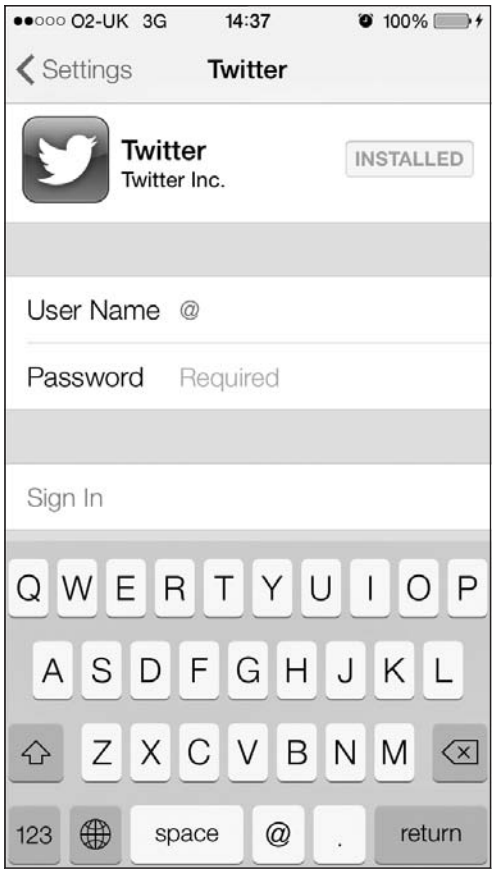


**Рис. 15.1.** Клавиатура на iPhone, книжная ориентация


**Рис. 15.2.** Клавиатура в iPhone, альбомная ориентация

Но на iPad клавиатура выглядит немного иначе. Самое очевидное отличие заключается в том, что эта клавиатура гораздо крупнее, чем на iPhone, поскольку сам экран у iPad шире. При альбомной ориентации клавиатура iPad значительно шире, но на ней содержится тот же набор клавиш, что и при книжной ориентации. Кроме того, пользователь может при желании «разбирать» клавиатуру iPad на части. Благодаря этому он лучше контролирует функционирование клавиатуры, зато появляются дополнительные проблемы у программистов, дизайнеров пользовательских интерфейсов и пользовательских взаимодействий.

iOS широковещательно распространяет различные уведомления, касающиеся отображения клавиатуры на экране. Вот список этих уведомлений и краткие характеристики каждого из них:

- `UIKeyboardWillShowNotification` — распространяется, когда клавиатура вот-вот появится на экране. Уведомление несет с собой словарь с пользовательской информацией, в котором содержатся различные данные о клавиатуре, анимации, которая будет применяться при выводе клавиатуры на экран, и другая информация;
- `UIKeyboardDidShowNotification` — распространяется, когда клавиатура уже появилась на экране;
- `UIKeyboardWillHideNotification` — распространяется, когда клавиатура вот-вот будет убрана с экрана. Уведомление несет с собой словарь с пользовательской информацией, в котором содержатся различные данные о клавиатуре, анимации, которая будет применяться при уходе клавиатуры с экрана, длительности анимации и т. д.;
- `UIKeyboardDidHideNotification` — распространяется после того, как клавиатура полностью скроется с экрана.

Уведомления `UIKeyboardWillShowNotification` и `UIKeyboardWillHideNotification` несут с собой словари с пользовательской информацией. В этих словарях содержатся валидные ключи и значения. Далее перечислены те из ключей, которые могут быть вам интересны:

- `UIKeyboardAnimationCurveUserInfoKey` — значение этого ключа характеризует тип анимационной кривой, которая будет использоваться при выводе клавиатуры на экран или ее скрытии. Этот ключ содержит значение типа `NSNumber` (инкапсулированное в объекте типа `NSValue`), которое, в свою очередь, содержит беззнаковое целое типа `NSUInteger`;
- `UIKeyboardAnimationDurationUserInfoKey` — значение данного ключа указывает в секундах длительность анимации, применяемой при отображении или скрытии клавиатуры. Если клавиатура вот-вот будет отображена, то это будет рамка, в которой появится клавиатура. Если клавиатура уже есть на экране, это будет контур, обрамляющий клавиатуру на экране перед тем, как она уйдет с экрана. Этот ключ содержит значение типа `CGRect` (инкапсулированное в объекте типа `NSValue`);
- `UIKeyboardFrameBeginUserInfoKey` — значение данного ключа указывает размеры рамки клавиатуры до того, как начнется анимация. Если клавиатура вот-вот отобразится, то перед появлением клавиатуры появится эта рамка. Если кла-

виатура в данный момент отображена и вот-вот уйдет с экрана, это будет рамка, фактически занимаемая клавиатурой на экране, прежде чем клавиатура уйдет с экрана в сопровождении соответствующей анимации. Этот ключ содержит значение типа `CGRect` (инкапсулированное в объект типа `NSValue`);

- `UIKeyboardFrameEndUserInfoKey` — значение данного ключа описывает контур клавиатуры, который оформится после того, как будет применена анимация. Если клавиатура вот-вот появится на экране, то она займет именно этот контур. Если клавиатура уходит с экрана, то именно этот контур от нее освободится. Этот ключ содержит значение типа `CGRect` (инкапсулированное в объекте типа `NSValue`).

Рассмотрим пример. Мы собираемся создать простое приложение с единственным видом. Это приложение будет работать только на iPhone. В нем будут отображаться вид с изображением и текстовое поле. Текстовое поле находится в нижней части экрана. Итак, когда пользователь дотрагивается до текстового поля, чтобы ввести в него некоторый текст, на экране всплывает виртуальная клавиатура, полностью заслоняющая текстовое поле. Наша задача — анимировать содержимое вида и перераспределить элементы так, чтобы все они оставались видимыми, даже если экран наполовину закрыт клавиатурой. В этом приложении мы будем использовать раскадровки. В контроллере вида заполним вид с изображением, поместив в него прокручивающийся вид, и поместим в этом прокручивающемся виде и вид с изображением, и текстовое поле (рис. 15.3).

Прокручивающийся вид в данном примере является родительским видом как для вида с изображением, так и для текстового поля. Он целиком заполняет пространство своего родительского вида.

Я уже прикрепил прокручивающийся вид, вид с изображением и текстовое поле, определенные в раскадровке, к файлу реализации контроллера вида, вот так:

```
#import "ViewController.h"

@interface ViewController () <UITextFieldDelegate>
@property (weak, nonatomic) IBOutlet UIScrollView *scrollView;
@property (weak, nonatomic) IBOutlet UITextField *textField;
@property (weak, nonatomic) IBOutlet UIImageView *imageView;
@end

@implementation ViewController

...
```

Теперь, когда аутлеты прикреплены к свойствам в контроллере нашего вида, можно приступить к слушанию клавиатурных уведомлений:

```
- (void) viewWillAppear:(BOOL)paramAnimated{
  [super viewWillAppear:paramAnimated];
  NSNotificationCenter *center = [NSNotificationCenter defaultCenter];

  [center addObserver:self selector:@selector(handleKeyboardWillShow:)
```

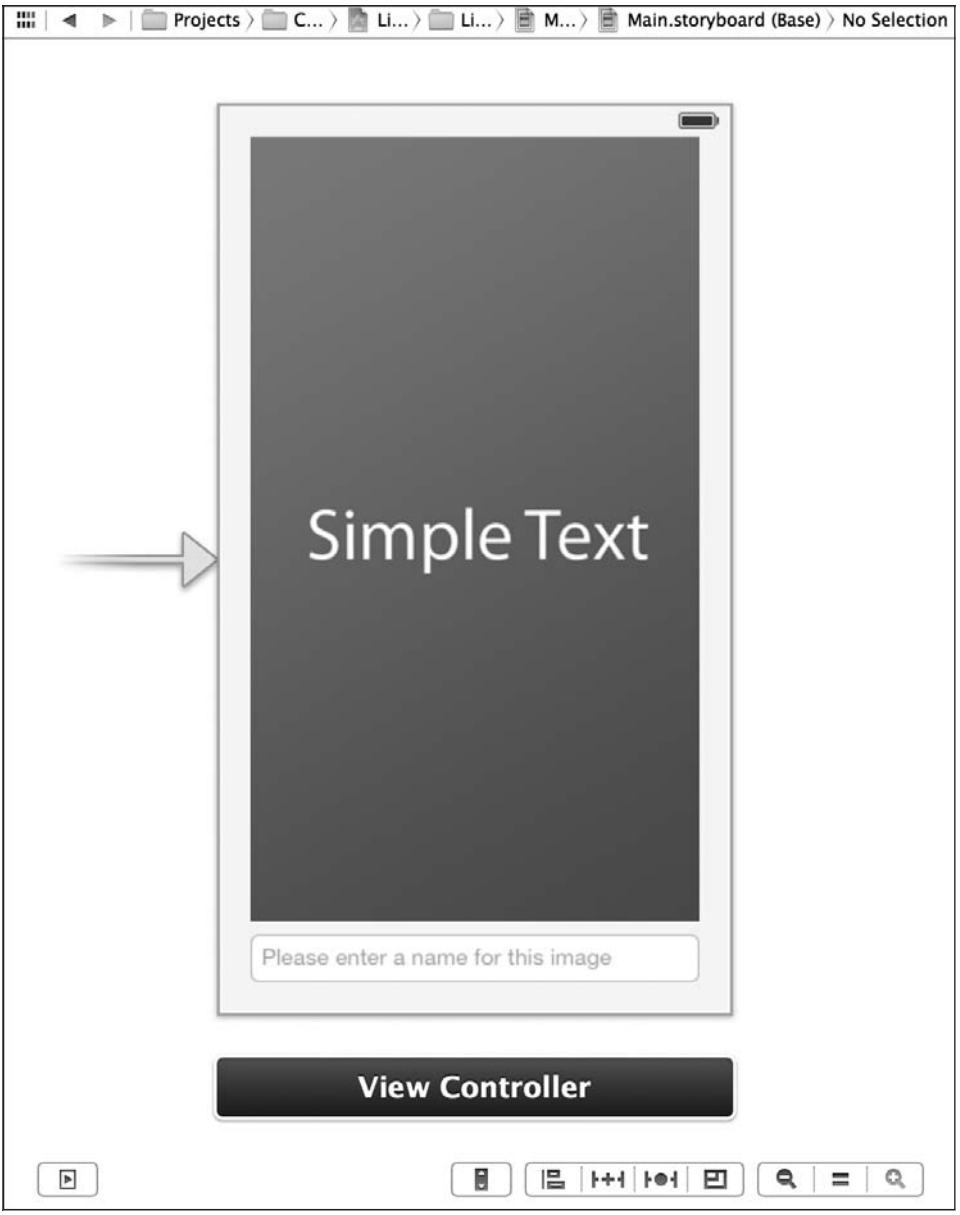


**Рис. 15.3.** Простая раскадровка, содержащая вид с изображением и текстовое поле

```
                name:UIKeyboardWillShowNotification object:nil];

  [center addObserver:self selector:@selector(handleKeyboardWillHide:)
                name:UIKeyboardWillHideNotification object:nil];
}
```

```
- (void)viewWillDisappear:(BOOL)paramAnimated{
  [super viewWillDisappear:paramAnimated];
  [[NSNotificationCenter defaultCenter] removeObserver:self];
}
```

Многие программисты допускают распространенную ошибку: продолжают слушать клавиатурные уведомления, когда контроллер вида не отображается на экране. Они начинают слушать уведомления в методе viewDidLoad, а удаляют элементы, действовавшие в качестве наблюдателей, только в методе dealloc. Такой подход проблематичен, так как, когда наш вид не отображается на экране, а клавиатура вновь открывается в каком-то другом виде, вы не должны корректировать положение каких-либо других компонентов контроллера вида. Учтите, что клавиатурные уведомления, как и любые другие, широковещательно передаются всем объектам-наблюдателям в контексте вашего приложения. Поэтому придется принимать дополнительные меры, чтобы программа реагировала не на все клавиатурные уведомления. Если клавиатурные уведомления поступают от вида, не отображаемого на экране в данный момент, они должны игнорироваться.

В предыдущем фрагменте кода мы начали слушать уведомления типа «клавиатура будет отображена» в методе экземпляра `handleKeyboardWillShow:` контроллера нашего вида. Уведомления типа «клавиатура будет скрыта» мы ожидаем в методе `handleKeyboardWillHide:`. Пока эти методы еще не написаны. Начнем с первого метода, `handleKeyboardWillShow:`. В этом методе нам требуется определить высоту клавиатуры, воспользовавшись ключом `UIKeyboardFrameEndUserInfoKey` из словаря с пользовательской информацией, сопровождающего уведомление. Это значение мы используем, чтобы переместить содержимое вида вверх — так, чтобы все необходимые элементы оказались над клавиатурой. Здесь приятно вспомнить, что мы поместили все нужное содержимое в прокручивающемся виде. Соответственно, потребуется всего лишь откорректировать краевые отступы прокручивающегося вида:

```
- (void) handleKeyboardWillShow:(NSNotification *)paramNotification{

  NSDictionary *userInfo = paramNotification.userInfo;

  /* Получаем длительность клавиатурной анимации — время, за которое
   клавиатура успеет отобразиться на экране. При анимировании и перемещении
   содержимого вида мы будем применять такое же значение длительности.
  */
  NSValue *animationDurationObject =
  userInfo[UIKeyboardAnimationDurationUserInfoKey];

  NSValue *keyboardEndRectObject = userInfo[UIKeyboardFrameEndUserInfoKey];
  double animationDuration = 0.0;
  CGRect keyboardEndRect = CGRectMake(0.0f, 0.0f, 0.0f, 0.0f);

  [animationDurationObject getValue:&animationDuration];
  [keyboardEndRectObject getValue:&keyboardEndRect];

  UIWindow *window = [UIApplication sharedApplication].keyWindow;
```

```
  /* Переводим размеры контура клавиатуры из координатной системы окна
   в координатную систему нашего вида. */
  keyboardEndRect = [self.view convertRect:keyboardEndRect
                    fromView:window];

  /* Определяем, в какой степени наш вид накрыт клавиатурой */
  CGRect intersectionOfKeyboardRectAndWindowRect =
  CGRectIntersection(self.view.frame, keyboardEndRect);

  /* Прокручиваем прокручивающийся вид таким образом, чтобы содержимое
   нашего вида отображалось полностью */
  [UIView animateWithDuration:animationDuration animations:^{

    self.scrollView.contentInset =
    UIEdgeInsetsMake(0.0f,
                     0.0f,
                     intersectionOfKeyboardRectAndWindowRect.size.height,
                     0.0f);

    [self.scrollView scrollRectToVisible:self.textField.frame animated:NO];

  }];

}
```

У нас получился довольно интересный и прямолинейный код. Единственная деталь, возможно требующая дополнительного разъяснения, — это функция `CGRectIntersection`. В ней мы получаем информацию о прямоугольном контуре клавиатуры (о верхней границе, левой границе, ширине и высоте). Это параметры клавиатуры в момент завершения анимации, когда она полностью отобразится на экране. Теперь, зная параметры клавиатуры, можем воспользоваться функцией `CGRectIntersection` и определить, какая часть нашего вида накрыта клавиатурой. Итак, берем контур клавиатуры, контур вида, а затем определяем, какая часть контура вида накрыта контуром клавиатуры. В результате получаем структуру типа `CGRect`, соответствующую той прямоугольной области вида, которая накрыта клавиатурой. Известно, что клавиатура появляется на нижней границе экрана и в ходе анимации выплывает вверх. Поэтому нас интересует вертикаль этой области. Итак, мы получаем высоту области пересечения контура клавиатуры и контура вида, а затем поднимаем на эту высоту содержимое вида. Длительность анимации перемещения задаем равной длительности анимации выдвижения клавиатуры. Таким образом, движения клавиатуры и поднимающихся экранных элементов синхронизируются.

Далее нужно написать метод `handleKeyboardWillHide:`. В нем мы будем скрывать клавиатуру — соответственно, она больше не будет закрывать наш вид. Итак, в этом методе всего лишь требуется сбросить размеры краевых отступов прокручивающегося вида к начальным значениям, перенести все элементы обратно вниз, чтобы вид выглядел точно так же, как было до появления клавиатуры:

```
- (void) handleKeyboardWillHide:(NSNotification *)paramSender{

  NSDictionary *userInfo = [paramSender userInfo];

  NSValue *animationDurationObject =
  [userInfo valueForKey:UIKeyboardAnimationDurationUserInfoKey];

  double animationDuration = 0.0;

  [animationDurationObject getValue:&animationDuration];

  [UIView animateWithDuration:animationDuration animations:^{
    self.scrollView.contentInset = UIEdgeInsetsZero;
  }];

}
```

И последний важный момент. Поскольку наш контроллер вида является делегатом текстового поля, необходимо обеспечить уход клавиатуры с экрана, если пользователь нажимает клавишу Return (Ввод) после ввода той или иной информации в текстовое поле:

```
- (BOOL) textFieldShouldReturn:(UITextField *)paramTextField{
  [paramTextField resignFirstResponder];

  return YES;
}
```

## См. также

Разделы 1.19 и 1.20.

# 15.4. Планирование локальных уведомлений

## Постановка задачи

Вы разрабатываете приложение, оперирующее данными о времени, например программу-будильник или программу-календарь. Это приложение должно информировать пользователя о событии в определенный момент времени, даже если в данный момент это приложение работает в фоновом режиме или вообще не запущено.

## Решение

Инстанцируйте объект типа `UILocalNotification`, сконфигурируйте его (далее будет рассказано о том, как это делается) и запланируйте с помощью метода экземпляра

scheduleLocalNotification:, относящегося к классу UIApplication. Чтобы получить экземпляр объекта вашего приложения, воспользуйтесь методом класса sharedApplication, относящимся к классу UIApplication.

## Обсуждение

Если в данный момент приложение работает в фоновом режиме или не работает вообще, система выдает пользователю так называемое *локальное уведомление*. Чтобы запланировать доставку локального уведомления, используется метод экземпляра scheduleLocalNotification:, относящийся к классу UIApplication. Если приложение работает *в приоритетном режиме* и в это время срабатывает запланированное локальное уведомление, пользователь не получает никакого оповещения об этом. Вместо этого iOS бесшумно дает вам знать о том, что было выдано уведомление, — это делается через делегат приложения. Пока не будем вдаваться в детали этого процесса, рассмотрим его чуть позже.

Можно приказать iOS доставить локальное уведомление пользователю когда-нибудь в будущем, когда ваше приложение даже не будет работать. Кроме того, такие уведомления могут быть периодическими, например запускаться каждую неделю в определенное время. При этом необходимо особенно внимательно указывать *дату запуска* (Fire Date) ваших уведомлений.

Метод экземпляра cancelAllLocalNotifications отменяет доставку всех стоящих в очереди локальных уведомлений, поступивших от вашего приложения.

Уведомление типа UILocalNotification имеет много свойств. Наиболее важными из них являются следующие:

- fireDate — это свойство типа NSDate, сообщающее iOS, когда должен быть запущен экземпляр локального уведомления. Данное свойство является обязательным;
- timeZone — это свойство типа NSTimeZone сообщает iOS, к какому часовому поясу относится конкретная дата запуска. Для получения актуального часового пояса используется метод экземпляра timeZone, относящийся к классу NSCalendar. Вы можете получить актуальный календарь, воспользовавшись методом класса currentCalendar, относящимся к вышеупомянутому классу;
- alertBody — это свойство относится к типу NSString и задает текст, который должен выводиться для пользователя при отображении вашего уведомления на экране;
- hasAction — логическое свойство. Сообщает iOS, собирается ли ваше приложение предпринимать какое-либо действие, когда происходит уведомление. Если установить его в YES, iOS отобразит для пользователя диалоговое окно, указанное в свойстве alertAction (описано далее). Если установить его в NO, iOS выдаст пользователю диалоговое окно с простым сообщением о том, что пришло уведомление;
- alertAction — если свойство hasAction установлено в YES, то значением этого свойства должна быть локализованная строка, описывающая действие, которое

пользователь может совершить над вашим уведомлением в случаях, когда уведомление произошло, но в этот момент приложение не работает в приоритетном режиме. После этого iOS выведет уведомление в центре уведомлений или на экране блокировки. Если свойство `hasAction` имеет значение `NO`, то свойство `alertAction` должно иметь значение `nil`;

- `applicationIconBadgeNumber` — если при срабатывании данного уведомления обязательно должен измениться номер ярлыка вашего приложения, то в этом свойстве можно задать желаемый номер. Значением этого свойства всегда является целое число. Когда вы присваиваете новое значение этому свойству, оно, как правило, должно представлять собой актуальный номер ярлыка вашего приложения плюс 1. Чтобы узнать текущий номер ярлыка приложения, пользуйтесь свойством `applicationIconBadgeNumber` класса `UIApplication`;
- `userInfo` — это экземпляр словаря `NSDictionary`, прикрепляемый к вашему уведомлению и получаемый приложением при доставке этого уведомления. Обычно такие словари используются для сообщения дополнительной информации о локальном уведомлении.

Благодаря суммарному эффекту свойств `hasAction` и `alertAction` пользователь может жестом смахивания запустить ваше уведомление в центре уведомлений. После этого iOS откроет приложение. Именно так пользователь может воздействовать на локальные уведомления. Это очень удобно, особенно если вы разрабатываете приложение-календарь. В таком приложении вы можете выдавать пользователю локальное уведомление за несколько дней до наступления дня рождения друга этого пользователя. Затем вы предоставляете пользователю возможность выполнить какую-то операцию над этим уведомлением. Например, когда пользователь открывает ваше приложение, вы можете предложить на выбор несколько виртуальных подарков, которые можно рассылать друзьям ко дню рождения.

Предположим, что в Лондоне сейчас 13:00, лондонец работает с вашим приложением на своем устройстве. Очевидно, что он находится в гринвичском часовом поясе (GMT + 0). Вы хотите доставить пользователю определенное уведомление в 14:00, даже если в этот момент ваше приложение не будет работать. И вот наш пользователь садится на самолет в лондонском аэропорту Гэтвик и собирается лететь в Стокгольм, то есть в часовой пояс GMT + 1. Допустим, полет длится полчаса. Тогда пользователь окажется в Стокгольме в 13:30 по лондонскому времени. Но когда самолет приземлится, iOS обнаружит, что часовой пояс изменился, и время на пользовательском устройстве также изменится — теперь будет 14:30. Вслед за этим iOS обнаружит, что необходимо отобразить уведомление (и так уже с опозданием на 30 минут, поскольку изменился часовой пояс), и отобразит его.

Проблема заключается в том, что ваше уведомление должно было быть отображено в 14:00 по времени GMT + 0 или в 15:00 по времени GMT + 1, но не в 14:30 по GMT + 1. Чтобы избежать подобных ситуаций (а ведь они довольно часты при нынешнем темпе жизни), при указании даты и времени для вывода на экран локальных уведомлений нужно также сообщать часовой пояс.

Теперь все это протестируем на практике. Напишем простое приложение, доставляющее локальное уведомление через 8 секунд после того, как пользователь впервые открывает это приложение:

```
#import "AppDelegate.h"

@implementation AppDelegate

- (BOOL) application:(UIApplication *)application
didFinishLaunchingWithOptions:(NSDictionary *)launchOptions{

  UILocalNotification *notification = [[UILocalNotification alloc] init];

  /* Настройки времени и часового пояса */
  notification.fireDate = [NSDate dateWithTimeIntervalSinceNow:8.0];
  notification.timeZone = [[NSCalendar currentCalendar] timeZone];

  notification.alertBody =
  NSLocalizedString(@"A new item is downloaded.", nil);

  /* Настройки действий */
  notification.hasAction = YES;
  notification.alertAction = NSLocalizedString(@"View", nil);

  /* Настройки ярлыка */
  notification.applicationIconBadgeNumber =
  [UIApplication sharedApplication].applicationIconBadgeNumber + 1;

  /* Дополнительная информация, пользовательский словарь */
  notification.userInfo = @{@"Key 1" : @"Value 1",
                            @"Key 2" : @"Value 2"};

  /* Назначаем уведомление */
  [[UIApplication sharedApplication] scheduleLocalNotification:notification];

  self.window = [[UIWindow alloc]
                 initWithFrame:[[UIScreen mainScreen] bounds]];
  self.window.backgroundColor = [UIColor whiteColor];
  [self.window makeKeyAndVisible];
  return YES;
}
```

Все это хорошо, но локальные уведомления практически бесполезны, пока мы не умеем на них реагировать и обрабатывать их при срабатывании. В разделе 15.5 подробнее рассказано об обработке таких уведомлений.

## См. также

Раздел 15.0.

# 15.5. Слушание локальных уведомлений и реагирование на них

## Постановка задачи

Вы научились планировать локальные уведомления (см. раздел 15.4). При поступлении этих уведомлений в приложение на них нужно правильно реагировать.

## Решение

Реализуйте метод `application:didReceiveLocalNotification:` делегата вашего приложения и считайте ключ `UIApplicationLaunchOptionsLocalNotificationKey`, относящийся к словарю параметров запуска вашего приложения при вызове метода `application:didFinishLaunchingWithOptions:` в делегате приложения. В подразделе «Обсуждение» данного раздела подробнее объяснено, почему приходится обрабатывать локальное уведомление в двух местах, а не в одном.

## Обсуждение

Когда происходит доставка локального уведомления и вам приходится его обрабатывать, приложение может находиться в одном из нескольких состояний. В зависимости от состояния обработка уведомления будет происходить по-разному. Вот ряд ситуаций, в которых iOS может доставить вашему приложению заранее запланированное локальное уведомление.

- В момент прихода локального уведомления приложение открыто и пользователь работает с ним. В таком случае при доставке уведомления вызывается метод `application:didReceiveLocalNotification:`.
- Локальное уведомление доставлено, но пользователь перевел приложение в фоновый режим. Как только пользователь дотрагивается до появившегося на экране уведомления, iOS может запустить приложение. В таком случае опять же вызывается метод `application:didReceiveLocalNotification:` делегата вашего приложения.
- В момент доставки локального уведомления приложение вообще неактивно. В данном случае вызывается метод `application:didFinishLaunchingWithOptions:` делегата приложения. Ключ `UIApplicationLaunchOptionsLocalNotificationKey` в словарном параметре `didFinishLaunchingWithOptions` этого метода содержит локальное уведомление, которое и привело к активизации приложения.
- Локальное уведомление поступает, когда пользовательское устройство заблокировано, независимо от состояния приложения: работает ли оно в приоритетном режиме, в фоновом режиме или вообще не работает. В таком случае приложение будет запущено одним из вышеупомянутых способов, зависящим от того, находилось ли ваше приложение в фоновом режиме, когда пользователь попытался открыть его через уведомление.

Разовьем код, рассмотренный в качестве примера в разделе 15.4. При запуске уведомления независимо от того, в каком состоянии в этот момент находится приложение, мы обработаем это уведомление (выведем для пользователя окно с предупреждением). Сначала используем код, изученный в разделе 15.4, в отдельном методе. Так мы сможем просто вызвать этот метод и назначить новое локальное уведомление. Вот почему поступаем именно так: в данном случае мы сможем посмотреть в центре уведомлений iOS, открылось ли приложение после того, как пользователь нажал появившееся на экране локальное уведомление. Если приложение открылось, то мы *не будем* запускать другое локальное уведомление. Однако если локальное уведомление не открыло наше приложение, то запланируем новое локальное уведомление. Далее приведен метод приложения, назначающий локальные уведомления, которые должны доставляться приложению через 8 секунд после вызова метода:

```
- (void) scheduleLocalNotification{

  UILocalNotification *notification = [[UILocalNotification alloc] init];

  /* Настройки времени и часового пояса */
  notification.fireDate = [NSDate dateWithTimeIntervalSinceNow:8.0];
  notification.timeZone = [[NSCalendar currentCalendar] timeZone];

  notification.alertBody =
  NSLocalizedString(@"A new item is downloaded.", nil);

  /* Настройки действий */
  notification.hasAction = YES;
  notification.alertAction = NSLocalizedString(@"View", nil);

  /* Настройки ярлыка */
  notification.applicationIconBadgeNumber =
  [UIApplication sharedApplication].applicationIconBadgeNumber + 1;

  /* Дополнительная информация, пользовательский словарь */
  notification.userInfo = @{@"Key 1" : @"Value 1",
                            @"Key 2" : @"Value 2"};

  /* Назначаем уведомление */
  [[UIApplication sharedApplication] scheduleLocalNotification:notification];

}
```

Метод, который мы здесь написали, называется `scheduleLocalNotification`. Как понятно из его названия, он просто создает объект уведомления и запрашивает iOS назначить это уведомление. Не путайте наш собственный метод `scheduleLocalNotification` с методом iOS, который называется `scheduleLocalNotification:` и относится к классу `UIApplication` (как и у всех методов iOS, в конце названия этого метода стоит двоеточие). Этот метод можно считать удобным вспомогательным инструментом, выполняющим сложную задачу назначения локального уведомления. При

этом сам он просто создает объект уведомления, а назначение этого уведомления делегирует iOS.

Теперь в методе `application:didFinishLaunchingWithOptions` мы проверим, открылось ли приложение именно по причине поступления имеющегося уведомления. Если это так, то будем работать с имеющимся локальным уведомлением. В противном случае назначим новое:

```
- (BOOL) application:(UIApplication *)application
didFinishLaunchingWithOptions:(NSDictionary *)launchOptions{
  if (launchOptions[UIApplicationLaunchOptionsLocalNotificationKey] != nil){
    UILocalNotification *notification =
    launchOptions[UIApplicationLaunchOptionsLocalNotificationKey];
    [self application:application didReceiveLocalNotification:notification];
  } else {
    [self scheduleLocalNotification];
  }

  self.window = [[UIWindow alloc]
                 initWithFrame:[[UIScreen mainScreen] bounds]];
  self.window.backgroundColor = [UIColor whiteColor];
  [self.window makeKeyAndVisible];
  return YES;
}
```

Когда в предыдущем коде наше приложение запускалось в результате поступления локального уведомления, мы перенаправляли локальное уведомление в метод `application:didReceiveLocalNotification:`, где оперировали имеющимся уведомлением и отображали для пользователя предупреждение. Вот простая реализация вышеупомянутого метода:

```
- (void) application:(UIApplication *)application
didReceiveLocalNotification:(UILocalNotification *)notification{

  NSString *key1Value = notification.userInfo[@"Key 1"];
  NSString *key2Value = notification.userInfo[@"Key 2"];

  if ([key1Value length] > 0 &&
    [key2Value length] > 0){

    UIAlertView *alert =
    [[UIAlertView alloc] initWithTitle:nil
                         message:@"Handling the local notification"
                         delegate:nil
                         cancelButtonTitle:@"OK"
                         otherButtonTitles:nil];
    [alert show];

  }

}
```

Теперь испытайте его. Опробуйте разные комбинации. Откройте приложение и поработайте с ним в приоритетном режиме, потом переведите его в фоновый режим, можете даже вообще закрыть. Посмотрите, как приложение функционирует в разных условиях.

## См. также

Разделы 15.0 и 15.4.

# 15.6. Обработка локальных системных уведомлений

## Постановка задачи

Когда ваше приложение возвращается в приоритетный режим, вам требуется возможность получать уведомления о важных системных изменениях, например об изменении локализации (языковых и культурных настроек) пользовательского устройства.

## Решение

Нужно просто слушать конкретное уведомление из числа тех, которые операционная система iOS посылает приложениям, переходящим в приоритетный режим. Далее перечислены некоторые из таких уведомлений:

- `NSCurrentLocaleDidChangeNotification` — доставляется приложениям, когда изменяется локализация устройства. Например, на странице Settings (Настройки) пользователь активизирует испанский язык вместо английского;
- `NSUserDefaultsDidChangeNotification` — запускается, когда пользователь изменяет настройки приложения на странице Settings (Настройки) устройства с iOS — при условии, что пользователь может изменить какую-либо настройку в вашем приложении;
- `UIDeviceBatteryStateDidChangeNotification` — запускается всякий раз, когда на устройстве с iOS изменяется состояние батареи. Например, если устройство подключается к компьютеру, когда приложение работает в приоритетном режиме, а потом отключается от компьютера, но приложение к этому моменту уже находится в фоновом режиме, то приложение получит такое уведомление (предполагается, что оно зарегистрировано на получение таких уведомлений). В подобном случае для считывания состояния можно узнать значение свойства `batteryState` экземпляра класса `UIDevice`;
- `UIDeviceProximityStateDidChangeNotification` — направляется приложению всякий раз, когда изменяется состояние датчика близости (Proximity Sensor). Последнее состояние можно узнать в свойстве `proximityState` экземпляра `UIDevice`.

## Обсуждение

Пока ваше приложение работает в фоновом режиме, может произойти многое! Например, пользователь может вдруг изменить локализацию устройства с iOS на странице Settings (Настройки) и задать, к примеру, испанский язык вместо английского. Приложения могут регистрироваться для получения таких уведомлений. Эти уведомления будут объединяться, а потом вместе доставляться приложению, переходящему в приоритетный режим. Объясню, что я понимаю в данном случае под объединением). Предположим, что ваше приложение работает в приоритетном режиме и вы зарегистрировали его для получения уведомлений `UIDeviceOrientationDidChangeNotification`. Вот пользователь нажимает кнопку Home (Домой), и ваше приложение уходит в фоновый режим. Потом пользователь изменяет ориентацию устройства с книжной на альбомную правую, затем возвращает книжную ориентацию и, наконец, переводит в альбомную левую. И когда пользователь вернет ваше приложение в приоритетный режим, оно получит всего одно уведомление типа `UIDeviceOrientationDidChangeNotification`. Это и есть объединение. Все остальные изменения ориентации, которые происходили, пока ваше приложение не было открыто, игнорируются (действительно, они не имеют значения, раз программы не было на экране, когда они происходили), и система не будет сообщать информацию о них вашему приложению. Тем не менее система доставит вам как минимум одно уведомление по каждому аспекту, связанному с устройством, — в частности, по ориентации. Так вы сможете узнать самую актуальную информацию о том, в каком положении находится устройство.

Вот реализация простого контроллера вида, в котором эта техника используется для определения изменений ориентации:

```
#import "ViewController.h"

@implementation ViewController

- (void) orientationChanged:(NSNotification *)paramNotification{
  NSLog(@"Orientation Changed");
}

- (void)viewDidAppear:(BOOL)paramAnimated{

  [super viewDidAppear:paramAnimated];

  /* Слушаем уведомление */
  [[NSNotificationCenter defaultCenter]
   addObserver:self
   selector:@selector(orientationChanged:)
   name:UIDeviceOrientationDidChangeNotification
   object:nil];
}

- (void) viewDidDisappear:(BOOL)paramAnimated{
  [super viewDidDisappear:paramAnimated];

  /* Прекращаем слушать уведомление */
```

```
  [[NSNotificationCenter defaultCenter]
   removeObserver:self
   name:UIDeviceOrientationDidChangeNotification
   object:nil];
}
@end
```

Теперь запустите приложение на устройстве. После того как на экране отобразится контроллер вида, нажмите кнопку Home (Домой) для перевода приложения в фоновый режим. После этого попробуйте пару раз изменить ориентацию устройства, а потом перезапустите приложение. Просмотрите результаты и обратите внимание на то, что, когда приложение открывается, обычно направляется одно уведомление к методу `orientationChanged:`.

Теперь допустим, что в вашем приложении пользователю предоставляется пакет с настройками. Как только приложение возвращается в приоритетный режим, требуется получать уведомления о тех изменениях, которые пользователь внес в настройки программы (пока приложение было в фоновом режиме).

## Решение

Зарегистрируйтесь для получения уведомлений `NSUserDefaultsDidChangeNotification`.

## Обсуждение

В приложениях, написанных для iOS, файл пакета настроек может быть предоставлен пользователю для внесения собственных настроек. Эти настройки будут доступны пользователю в приложении (Settings) на устройстве. Чтобы лучше понять, как работает этот механизм, создадим пакет с настройками.

1. В Xcode выберите File ▸ New File (Файл ▸ Новый файл).
2. Убедитесь, что слева задана категория iOS.
3. Выберите подкатегорию Resources (Ресурсы).
4. В качестве типа файла укажите пакет настроек (Settings Bundle), а потом нажмите Next (Далее).
5. Назовите файл `Settings.bundle`.
6. Нажмите Save (Сохранить).

Итак, теперь у вас в Xcode есть файл под названием `Settings.bundle`. Оставьте этот файл как есть, не вносите в него никаких изменений. Нажмите кнопку Home (Домой) и перейдите в приложение Settings (Настройки). Если вы назовете свое приложение `foo`, то в окне настроек, показанном на рис. 15.4, также будет указано `Foo`. (Мое приложение я назвал `Handling local System Notifications`, это название вы видите на рисунке.)

Щелкните на имени приложения, чтобы просмотреть, какие настройки в приложении предоставляются пользователю. Нас интересует, когда пользователь вносит изменения в эти настройки, чтобы при необходимости мы могли соответствующим образом изменить внутреннее состояние приложения.

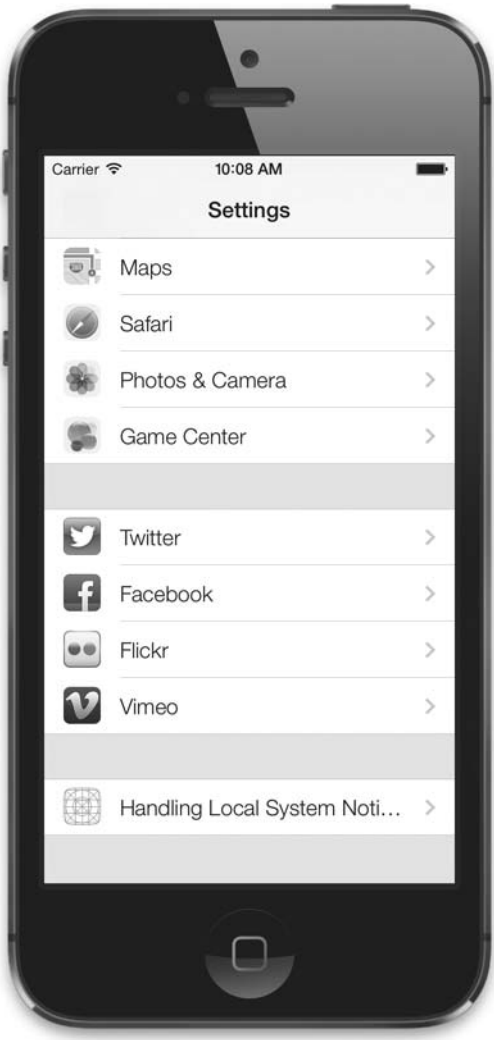


**Рис. 15.4.** Пакет Settings.bundle отображается в приложении Settings (Настройки) в симуляторе iOS

Далее начнем слушать в делегате нашего приложения уведомления `NSUserDefaultsDidChangeNotification`. Когда приложение завершится, мы, само собой, удалим делегат из цепочки адресатов уведомлений:

```
#import "AppDelegate.h"

@implementation AppDelegate

- (void) handleSettingsChanged:(NSNotification *)paramNotification{

  NSLog(@"Settings changed");
```

```
  NSLog(@"Notification Object = %@", paramNotification.object);

}
- (BOOL)            application:(UIApplication *)application
  didFinishLaunchingWithOptions:(NSDictionary *)launchOptions{

    [[NSNotificationCenter defaultCenter]
     addObserver:self
     selector:@selector(settingsChanged:)
     name:NSUserDefaultsDidChangeNotification
     object:nil];

    return YES;
}

- (void)applicationWillTerminate:(UIApplication *)application{
  [[NSNotificationCenter defaultCenter] removeObserver:self];
}
@end
```

А теперь попробуйте изменить некоторые из этих настроек, пока приложение работает в фоновом режиме. Когда закончите, переведите приложение в приоритетный режим — и увидите, что программе доставлено уведомление `NSUserDefaultsDidChangeNotification`. Объект, прикрепленный к этому уведомлению, будет относиться к типу `NSUserDefaults` и содержать настройки вашего приложения `user defaults`.

# 15.7. Настройка приложения для получения пуш-уведомлений

## Постановка задачи

Требуется сконфигурировать приложение таким образом, чтобы сервер мог по своей инициативе отправлять уведомления на различные устройства.

## Решение

Выполните следующие шаги.

1. Настройте профиль инициализации для вашего приложения, активизировав в нем возможность получения пуш-уведомлений.
2. В приложении зарегистрируйте устройство для получения пуш-уведомлений для этого приложения.
3. Возьмите идентификатор пуш-уведомлений данного устройства для вашего приложения и отошлите этот идентификатор на сервер.

В этом разделе мы поговорим о приложении, применяемом для внесения настроек и регистрации вашего приложения для получения пуш-уведомлений. Серверную часть этой технологии затрагивать не будем, этим вопросам будет посвящен отдельный раздел.

## Обсуждение

Пуш-уведомления похожи на локальные уведомления тем, что позволяют сообщать пользователю определенную информацию, даже если ваше приложение неактивно при поступлении уведомления. В то время как локальные уведомления назначаются самим приложением, пуш-уведомления конфигурируются на сервере и отсылаются с него в Apple, а Apple сама раздает эти уведомления на разные устройства, работающие по всему миру. Мы должны выполнять серверную часть работы, поэтому сами составляем пуш-уведомления и отправляем их на серверы APNS (Apple Push Notifications Services). Затем APNS пытается доставлять наши пуш-уведомления по защищенным каналам на устройства, которые зарегистрированы для получения таких уведомлений.

Чтобы приложение для iOS могли получать пуш-уведомления, у него должен быть валидный профиль инициализации, в котором активизирована возможность получения пуш-уведомлений. Чтобы правильно сконфигурировать профиль, выполните следующие шаги.

Предполагается, что вы уже настроили на портале разработчика ваши сертификаты для разработки и распространения приложения. Чтобы автоматически сконфигурировать сертификаты, можете воспользоваться новой настройкой Accounts (Учетные записи), появившейся в Xcode. Просто перейдите в раздел Xcode Preferences (Настройки), в нем откройте область Accounts (Учетные записи). Добавьте в список учетных записей ваш идентификатор Apple ID, а далее Xcode сконфигурирует сертификаты за вас.

1. Войдите в центр для разработки для iOS.
2. Перейдите в раздел Certificates, Identifiers & Profiles (Сертификаты, идентификаторы и профили) — он расположен справа.
3. В разделе Identifiers (Идентификаторы) создайте для себя новый идентификатор App ID. Он должен иметь валидный Explicit App ID (явный идентификатор приложения), например `com.pixolity.ios.cookbook.PushNotificationApp`. Обратите внимание: это имя построено по принципу обратной записи домена в стиле, который я выбрал для этого приложения. Выберите такой идентификатор приложения в стиле обратной записи домена, который подходит вам и вашей организации.
4. В разделе App Services (Сервисы приложения) на странице идентификатора приложения убедитесь, что установили флажок для пуш-уведомлений (Push Notifications) (рис. 15.5).
5. Как только вас устроит конфигурация идентификатора приложения (рис. 15.6), отправьте ваш App ID в Apple.
6. После того как настроите все детали конфигурации идентификатора приложения, перейдите в раздел Provisioning Profiles (Профили инициализации) вашего iOS-портала.
7. Создайте профиль инициализации для разработки (Development). Позже вы можете создать и другие профили — Ad Hoc и App Store, так что об этом не волнуйтесь. Когда будете готовы отправить ваше приложение в App Store, можете просто вернуться к этому шагу и сгенерировать профили Ad Hoc и App Store.

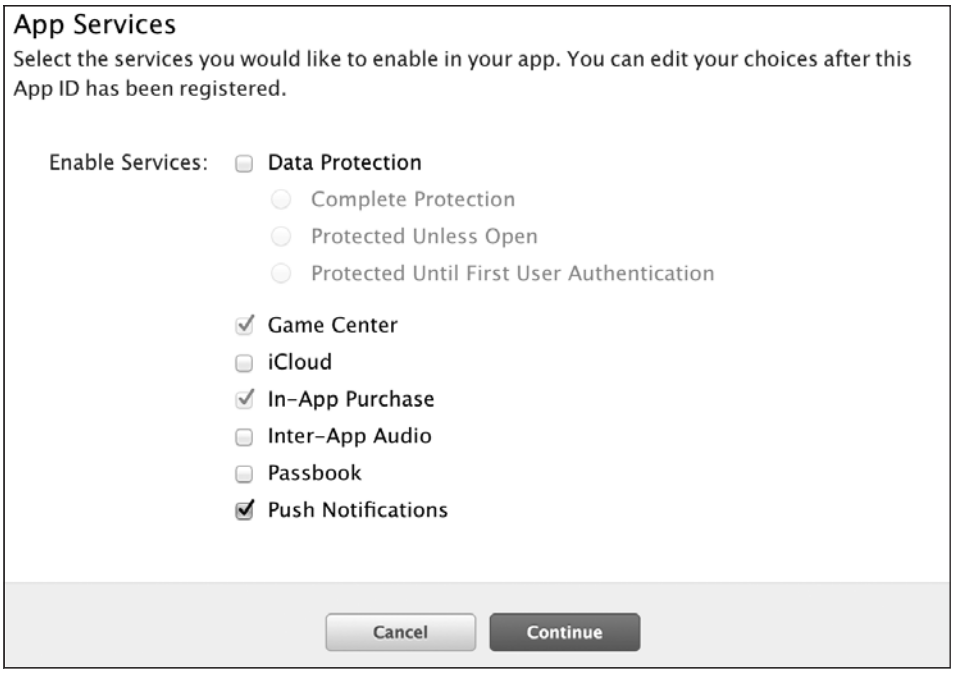


**Рис. 15.5.** Активизация пуш-уведомлений для App ID

8. Убедитесь, что новый профиль инициализации для разработки приложения связан с идентификатором приложения, который вы сгенерировали ранее. Это первый вопрос, который у вас попытается выяснить система при генерации профиля инициализации.
9. Когда профиль будет готов, скачайте его и перетащите в программу iTunes на своем компьютере, чтобы установить. При установке профиля не делайте двойных щелчков на нем. При двойном щелчке кнопкой мыши имя файла установленного профиля будет сброшено и заменено хеш-именем MD5, по которому файл очень сложно идентифицировать на диске. Если вы аккуратно перетащите профиль в iTunes, то iTunes установит профиль под его оригинальным именем.
10. В Xcode в настройках сборки вашего приложения просто выберите сборку в соответствии с тем профилем, который только что создали. Убедитесь, что при разработке (схема Development) вы используете именно этот профиль, а затем примените в схеме Release (Релиз) профиль App Store или Ad Hoc, созданием которых займетесь позже.
11. Перетащите ваш профиль инициализации в текстовый редактор для OS X — например, в TextEdit. Найдите в файле профиля ключ `Entitlements`. Весь этот раздел моего профиля инициализации выглядит так:

```
<key>Entitlements</key>
<dict>
  <key>application-identifier</key>
```

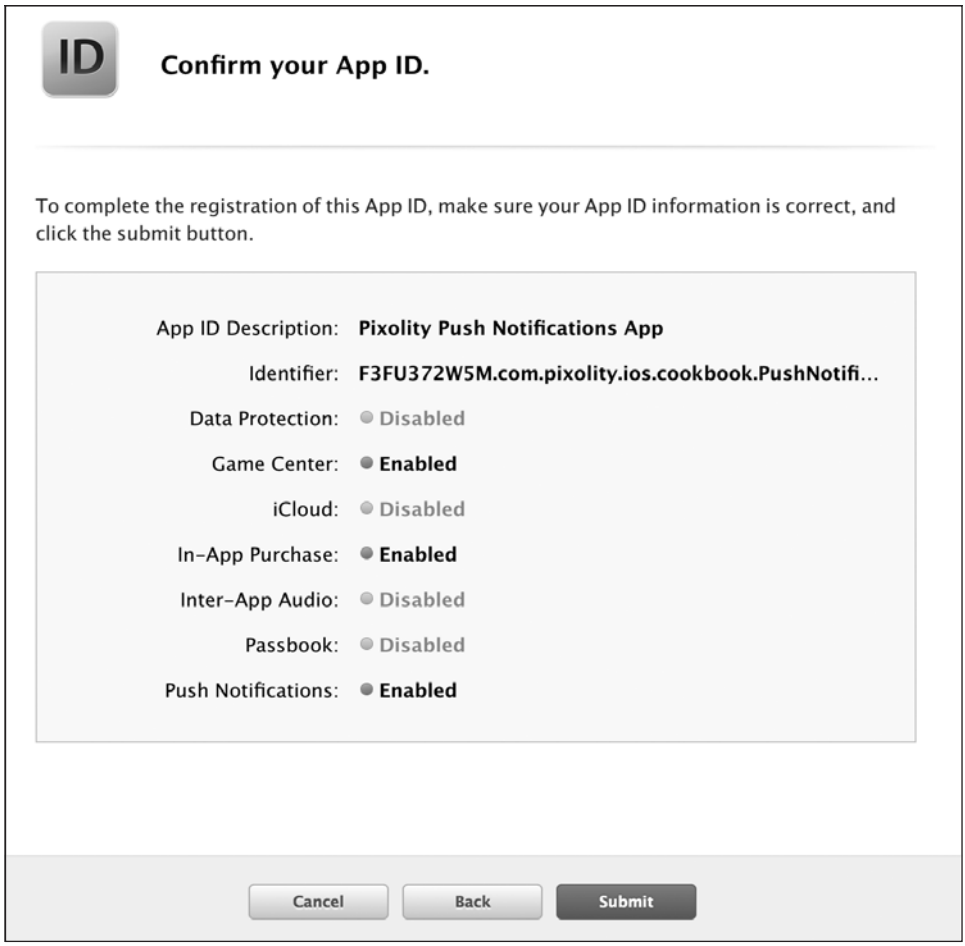


**Рис. 15.6.** Создание идентификатора приложения с поддержкой пуш-уведомлений

```
  <string>F3FU372W5M.com.pixolity.ios.cookbook.PushNotificationApp</string>
  <key>aps-environment</key>
  <string>development</string>
  <key>get-task-allow</key>
  <true/>
  <key>keychain-access-groups</key>
  <array>
    <string>F3FU372W5M.*</string>
  </array>
</dict>
```

12. Создайте новый PLIST-файл для вашего проекта в Xcode, назовите этот файл `Entitlements.plist`. В Xcode щелкните на этом файле правой кнопкой мыши, выберите Open As (Открыть как) и далее Source Code (Исходный код). Содержимое вашего файла изначально будет выглядеть вот так:

```
<plist version="1.0">
<dict/>
</plist>
```

13. Поместите разрешения вашего профиля инициализации прямо в файл `Entitlements.plist`, чтобы его содержимое выглядело вот так:

```
<plist version="1.0">
<dict>
  <key>application-identifier</key>
  <string>F3FU372W5M.com.pixolity.ios.cookbook.PushNotificationApp</string>
<key>aps-environment</key>
  <string>development</string>
  <key>get-task-allow</key>
  <true/>
  <key>keychain-access-groups</key>
  <array>
    <string>F3FU372W5M.*</string>
  </array>
</dict>
</plist>
```

Значения, приведенные в рассматриваемых здесь листингах, относятся к профилям, которые создал я. В вашем профиле будут другие значения и, конечно же, будет другой идентификатор приложения (App ID). Поэтому внимательно выполните предыдущие шаги, чтобы правильно создать идентификатор вашего приложения и профиль. Потом возьмите разрешения из своего профиля и вставьте их в файл Entitlements.plist вашего проекта.

14. Далее перейдите в настройки сборки вашего проекта. В разделе Code Signing Entitlements (Разрешения для подписания кода) введите значение `$(SRCROOT)/$(TARGET_NAME)/Entitlements.plist` (если создали файл разрешений в целевом каталоге проекта) или `$(SRCROOT)/Entitlements.plist` (если создали файл разрешений в корневом каталоге, включающем в себя весь исходный код). Если не знаете, какой вариант выбрать, просто попробуйте оба этих значения, попытайтесь собрать проект с каждым из них. Если Xcode сообщит, что не может найти файл разрешений, поставьте второе значение, и оно сработает. Настройка сборки Code Signing Entitlements (Разрешения для подписания кода) требует указания относительного пути, идущего к файлу разрешений от корневого каталога исходного кода. Поэтому, если вы поместите файл разрешений в какой-то другой каталог, вам потребуется выстроить этот путь вручную, а потом записать полученный путь в это поле.
15. Соберите проект и убедитесь, что Xcode не выдает никаких ошибок. Если получите сообщение о какой-либо ошибке, то, вероятно, дело в том, что вы задали неверный профиль инициализации либо указали в настройках сборки неверный путь к файлу разрешений для подписания кода.
16. В делегате вашего приложения вызовите метод `registerForRemoteNotificationTypes:`, относящийся к классу `UIApplication`, а затем передайте этому методу

значения UIRemoteNotificationTypeAlert, UIRemoteNotificationTypeBadge и UIRemoteNotificationTypeSound, как показано далее:

```
 - (BOOL) application:(UIApplication *)application
didFinishLaunchingWithOptions:(NSDictionary *)launchOptions{

  [[UIApplication sharedApplication] registerForRemoteNotificationTypes:
    UIRemoteNotificationTypeAlert |
    UIRemoteNotificationTypeBadge |
    UIRemoteNotificationTypeSound];

  self.window = [[UIWindow alloc]
                 initWithFrame:[[UIScreen mainScreen] bounds]];
  self.window.backgroundColor = [UIColor whiteColor];
  [self.window makeKeyAndVisible];
  return YES;
}
```

Так вы зарегистрируете ваше приложение для получения пуш-уведомлений. Такие уведомления могут содержать информацию с предупреждениями, изменения номера ярлыка вашего приложения, а также звуки. Пока не будем вдаваться в эти подробности. Просто зарегистрируйте приложение для получения пуш-уведомлений, как показано ранее. Как только вы это сделаете, iOS отошлет запрос о регистрации к APNS. Перед тем как это делать, iOS попросит у пользователя разрешения зарегистрировать данное приложение для получения пуш-уведомлений. Пользовательский интерфейс, открывающийся в iOS при этом запросе, показан на рис. 15.7.

17. Теперь реализуйте метод application:didRegisterForRemoteNotificationsWithDeviceToken: делегата вашего приложения. Этот метод вызывается, когда iOS удается успешно зарегистрировать устройство в APNS и присвоить ему маркер. Этот маркер действует только для конкретного приложения, установленного именно на данном устройстве.
18. Далее реализуйте метод application:didFailToRegisterForRemoteNotificationsWithError: делегата вашего приложения. Этот метод вызывается, если iOS не удается зарегистрировать приложение для получения пуш-уведомлений. Это может произойти потому, что ваш профиль настроен неправильно или устройство не подключено к Интернету, а также по многим другим причинам. В параметре didFailToRegisterForRemoteNotificationsWithError этого метода вы получите ошибку типа NSError. Ее можно проанализировать и узнать, по какой причине возникла проблема.

Теперь вы знаете все необходимое, чтобы настроить ваше приложение на получение пуш-уведомлений.

## См. также

Раздел 15.0.

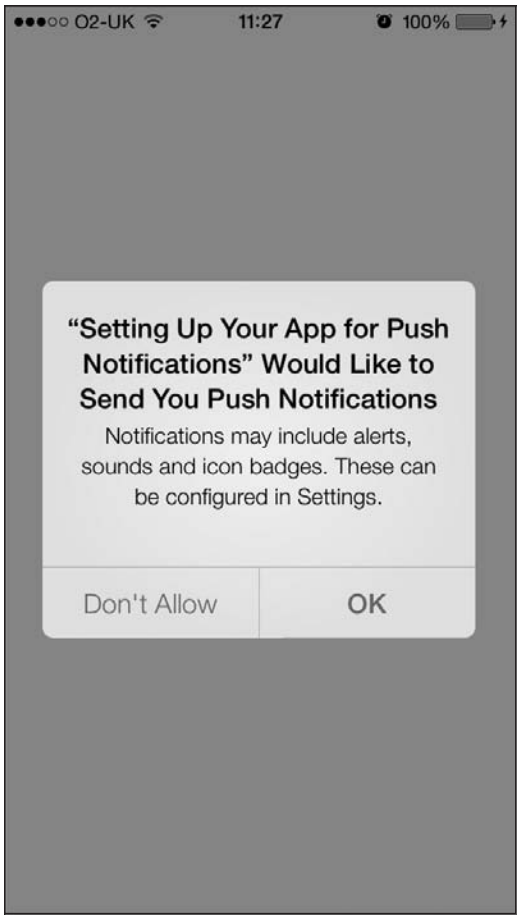


**Рис. 15.7.** iOS выдает пользователю запрос о разрешении на получение пуш-уведомлений

# 15.8. Доставка пуш-уведомлений в приложение

## Постановка задачи

Требуется отправлять пуш-уведомления на пользовательские устройства, которые зарегистрированы для получения таких уведомлений.

## Решение

Убедитесь, что вы собрали маркеры-идентификаторы пуш-уведомлений этих приложений (см. раздел 15.7). Затем сгенерируйте SSL-сертификаты, которые будут использоваться вашими веб-сервисами для отправки пуш-уведомлений на устрой-

ства. Затем создайте простой веб-сервис для отправки пуш-уведомлений на зарегистрированные устройства.

Этот материал — продолжение раздела 15.7. Обязательно прочтите предыдущий раздел и полностью в нем разберитесь, прежде чем переходить к изучению данного раздела.

## Обсуждение

Чтобы обмениваться информацией с серверами APNS, ваши веб-сервисы должны совершить акт *квитирования*[1] (handshaking). Это обмен сигналами с сервером, при котором используется выданный Apple SSL-сертификат. Чтобы сгенерировать такой сертификат, выполните следующие шаги.

1. Войдите в центр разработки для iOS.
2. Перейдите в раздел Certificates, Identifiers & Profiles (Сертификаты, идентификаторы, профили), расположенный справа.
3. В разделе идентификаторов приложений (App ID) найдите идентификатор вашего приложения, для которого задано получение пуш-уведомлений, выберите этот идентификатор и нажмите кнопку Settings (Настройки), чтобы его сконфигурировать, как показано на рис. 15.8.

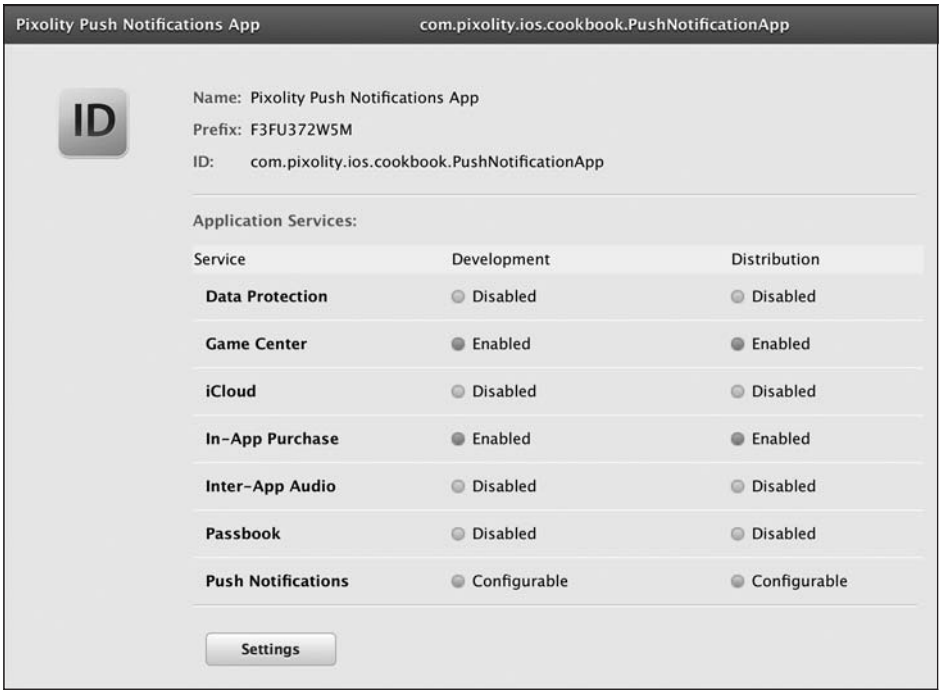


**Рис. 15.8.** Изменение настроек имеющегося идентификатора приложения

[1] Употребляется и буквальный перевод — «рукопожатие». — *Примеч. пер.*

4. В разделе настроек, называемом Push Notifications (Пуш-уведомления), найдите подраздел Development SSL Certificate (SSL-сертификат для разработки) и нажмите кнопку Create Certificate (Создать сертификат) (рис. 15.9). Далее следуйте указаниям Apple по созданию сертификата. Пока мы создаем SSL-сертификат для разработки приложения, поскольку на данном этапе нас интересует исключительно разработка. Позже, когда будем готовы отправить наше приложение в App Store, просто повторите подобный процесс и создайте SSL-сертификаты для распространения (Distribution).
5. Как только сертификат будет готов (рис. 15.10), скачайте его на компьютер и дважды щелкните на нем кнопкой мыши, чтобы импортировать его в вашу связку ключей.

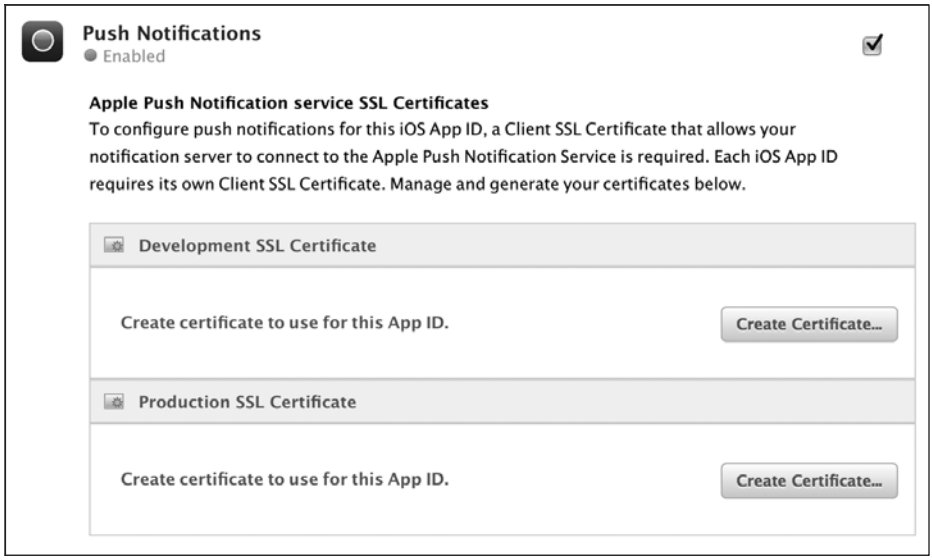


**Рис. 15.9.** Создание SSL-сертификата, используемого для операций с пуш-уведомлениями и разработки приложения

6. Теперь откройте окно Keychain Access (Доступ к связке ключей) на OS X и перейдите к связке ключей Login (если эта связка ключей задана у вас по умолчанию). В разделе My Certificates (Мои сертификаты) найдите тот сертификат, который вы только что импортировали в связку ключей, и раскройте его, щелкнув на маленькой кнопке со стрелкой слева. Так вы узнаете ассоциированный с этим приложением закрытый ключ (рис. 15.11).
7. Щелкните на сертификате правой кнопкой мыши и экспортируйте его как `.cer`-сертификат (а не как файл `.p12`). Назовите его `PushCertificate.cer`.
8. Щелкните правой кнопкой мыши на закрытом ключе и экспортируйте его как файл `.p12` (а не как файл сертификата). Назовите его `PushKey.p12`. Здесь потребуется указать пароль к закрытому ключу. Обязательно используйте такой пароль, который впоследствии сможете вспомнить.

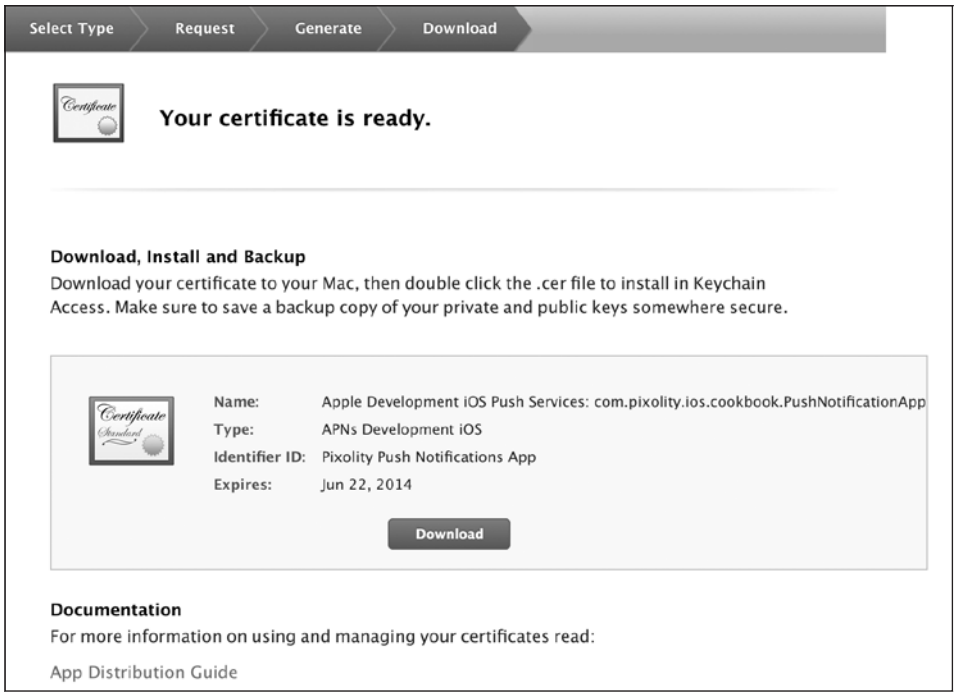


**Рис. 15.10.** SSL-сертификат для обмена информацией с APNS на этапе разработки приложения готов к скачиванию


**Рис. 15.11.** Сертификат для работы с пуш-уведомлениями на этапе разработки и его закрытый ключ

Отлично. Теперь, чтобы упростить дальнейший материал этого раздела, перейдем к использованию PHP. Применим этот язык для отправки на наше устройство

простого пуш-уведомления. В разделе 15.7 мы уже настроили получение пуш-уведомлений в этом приложении. Поскольку настройка использования PHP на сервере Apache не относится к темам этой книги, мы воспользуемся упрощенным вариантом и применим MAMP. MAMP установит у вас на компьютере Apache и PHP, если вы еще не сделали этого. На сайте MAMP даются подробные инструкции о том, какие шаги потребуется выполнить. Когда вы установите MAMP, корневой каталог со всеми вашими PHP-файлами будет находиться по адресу /Applications/MAMP/htdocs/. Если при последующей установке этот каталог изменится, откройте MAMP, перейдите в раздел Preferences (Настройки), а потом — к Apache. Вы найдете там и корневой каталог Apache.

Чтобы наш PHP-сценарий мог обмениваться информацией с APNS, потребуется задать ему SSL-сертификат, который мы сгенерировали ранее, на портале разработки для iOS. Именно поэтому мы извлекли `.cer`-файл сертификата и файл `.p12` закрытого ключа — теперь мы должны задать эти файлы нашему PHP-сценарию. Чтобы это сработало, используем в окне терминала `openssl`, а затем комбинируем сертификат и закрытый ключ `.p12` в едином PEM-файле. В этой книге мы не можем подробно поговорить о PEM-файлах — это тема для отдельной книги. Однако вы можете подробнее познакомиться с этой темой в документе RFC 1421.

Для создания PEM-файла выполните следующие шаги (предполагается, что вы уже экспортировали файлы `PushKey.p12` и `PushCertificate.cer` на ПК, как было описано ранее).

1. Откройте окно терминала (Terminal) в OS X. Введите в окне терминала следующую команду:

   ```
   openssl x509 -in PushCertificate.cer -inform der -out PushCertificate.pem
   ```

2. Чтобы преобразовать файл `.p12` в PEM-файл, введите в окне терминала следующую команду:

   ```
   openssl pkcs12 -nocerts -in PushKey.p12 -out PushKey.pem
   ```

3. Система потребует ввести пароль, который вы задали для этого закрытого ключа, когда экспортировали его из раздела Keychain Access. После того как пароль для импорта будет проверен и подтвержден, OpenSSL запросит у вас фразу-пароль для результирующего PEM-файла. Этот пароль должен содержать не менее четырех символов. Задайте такой пароль и хорошо запомните его для последующего использования.

4. Теперь у вас на ПК должно быть два PEM-файла: `PushCertificate.pem` и `PushKey.pem`. Нужно сложить их в единый PEM-файл — этот формат распознается PHP. Для этого воспользуйтесь следующей командой:

   ```
   cat PushCertificate.pem PushKey.pem > PushCertificateAndKey.pem
   ```

5. Теперь проверим, сможем ли мы подключиться к песочнице (речь идет о тестовой версии, только для целей разработки) с помощью сгенерированных нами PEM-файлов. В качестве песочницы используется защищенный APNS-сервер. Выполните в окне терминала следующую команду:

   ```
   openssl s_client -connect gateway.sandbox.push.apple.com:2195 \
   -cert PushCertificate.pem -key PushKey.pem
   ```

Если все будет нормально, то на данном этапе потребуется ввести фразу-пароль для вашего закрытого ключа. Помните ее? Хорошо, вводите. Если соединение будет успешно установлено, откроется OpenSSL, ожидающий от вас нескольких символов в качестве ввода. После получения этого ввода соединение будет закрыто. Введите любые случайные символы и нажмите Enter (Ввод). Соединение закрыто. Вам удалось успешно связаться с APNS-сервером, воспользовавшись вашим сертификатом и закрытым ключом.

Теперь напишем несложный PHP-сценарий для отправки простого пуш-уведомления на устройство. Но прежде чем мы двинемся дальше, необходимо получить маркер пуш-уведомлений, действующий на нашем устройстве. Мы должны получить его в таком формате, который понятен PHP. iOS инкапсулирует маркер пуш-уведомления в экземпляр `NSData`, но PHP неизвестно, что такое `NSData`. Нам нужно преобразовать этот маркер в строку, которую мы сможем использовать в нашем PHP-сценарии. Для этого считываем весь маркер байт за байтом и преобразуем каждый байт в шестнадцатеричное строковое представление:

```
- (void) application:(UIApplication *)application
didRegisterForRemoteNotificationsWithDeviceToken:(NSData *)deviceToken{

  /* Каждый байт данных будет преобразован в свое шестнадцатеричное
   значение, например 0x01 или 0xAB. При этом часть 0x отбрасывается.
   Итак, для представления 1 байта нам потребуются два символа, поэтому
   здесь указано * 2 */
  NSMutableString *tokenAsString = [[NSMutableString alloc]
 initWithCapacity:deviceToken.length * 2];

  char *bytes = malloc(deviceToken.length);
  [deviceToken getBytes:bytes];

  for (NSUInteger byteCounter = 0;
  byteCounter < deviceToken.length;
  byteCounter++){

    char byte = bytes[byteCounter];
    [tokenAsString appendFormat:@"%02hhX", byte];
  }

  free(bytes);

  NSLog(@"Token = %@", tokenAsString);

}
```

Запустите ваше приложение и убедитесь, что маркер устройства выводится на консоль, вот так:

```
Token = 05924634A8EB6B84437A1E8CE02E6BE6683DEC83FB38680A7DFD6A04C6CC586E
```

Отметьте себе этот маркер устройства, так как мы будем пользоваться им в PHP-сценарии:

```
<?php
  /* При разработке мы пользуемся защищенной версией APNS. При подготовке
   приложения для использования в реальных условиях измените это значение
   на ssl://gateway.push.apple.com:2195 */
  $apnsServer = 'ssl://gateway.sandbox.push.apple.com:2195';

  /* Убедитесь, что это значение совпадает с паролем, который вы задали
   для закрытого ключа при экспорте в .pem-файл, когда использовали openssl
   в системе OS X */
  $privateKeyPassword = '1234';

  /* Если хотите, запишите здесь собственное сообщение */
  $message = 'Welcome to iOS 7 Push Notifications';

  /* Запишите здесь маркер вашего устройства */
  $deviceToken =
    '05924634A8EB6B84437A1E8CE02E6BE6683DEC83FB38680A7DFD6A04C6CC586E';

  /* Замените эту информацию именем файла, указанного в файле вашего
   сценария. В этом файле должны содержаться сгенерированные вами ранее
   сертификат и закрытый ключ */
  $pushCertAndKeyPemFile = 'PushCertificateAndKey.pem';

  $stream = stream_context_create();

  stream_context_set_option($stream,
                            'ssl',
                            'passphrase',
                            $privateKeyPassword);

stream_context_set_option($stream,
                          'ssl',
                          'local_cert',
                          $pushCertAndKeyPemFile);

  $connectionTimeout = 20;
  $connectionType = STREAM_CLIENT_CONNECT | STREAM_CLIENT_PERSISTENT;
  $connection = stream_socket_client($apnsServer,
                                     $errorNumber,
                                     $errorString,
                                     $connectionTimeout,
                                     $connectionType,
                                     $stream);

  if (!$connection){
    echo "Failed to connect to the APNS server. Error no = $errorNumber<br/>";
    exit;
  } else {
    echo "Successfully connected to the APNS. Processing...</br>";
```

```
  }
  $messageBody['aps'] = array('alert' => $message,
                              'sound' => 'default',
                              'badge' => 2,
  );

  $payload = json_encode($messageBody);

  $notification = chr(0) .
                  pack('n', 32) .
                  pack('H*', $deviceToken) .
                  pack('n', strlen($payload)) .
  $payload;

  $wroteSuccessfully = fwrite($connection, $notification,
                       strlen($notification));

  if (!$wroteSuccessfully){
    echo "Could not send the message<br/>";
  }
  else {
    echo "Successfully sent the message<br/>";
  }

fclose($connection);
```

Если вы даже не программируете на PHP, внимательно просмотрите этот сценарий и почитайте комментарии к нему. Обязательно замените все значения в этом сценарии теми, что соответствуют вашему приложению. Например, используемый здесь маркер относится к моему устройству. Пользуйтесь маркером своего устройства, который выяснили ранее в этом разделе. У вас будут иные пароли, `.pem`-файлы, скорее всего, будут находиться в других местах. Ради дополнительного упрощения этого раздела я поместил свой PHP-сценарий в тот же каталог, в котором ранее сохранил закрытый ключ и `.pem`-файл сертификата (`PushCertificateAndKey.pem`). Поэтому я могу обращаться к `.pem`-файлу просто по его имени.

Если вы правильно выполнили все шаги и инструкции, описанные в этом разделе, то ваш PHP-сценарий должен открываться в браузере. После этого на устройство начнут поступать уведомления. Сценарий посылает уведомление на APNS-сервер, который уже доставляет это уведомление на устройство. Когда пуш-уведомление попадает на устройство (предполагается, что на устройстве в этот момент отображается экран блокировки), вы увидите на экране примерно такую картинку, какая показана на рис. 15.12.

## См. также

Раздел 15.7.

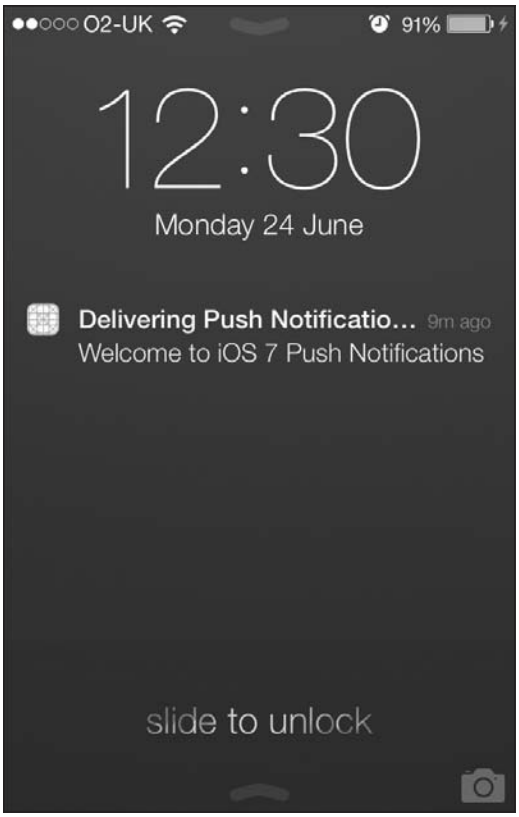


**Рис. 15.12.** Пуш-уведомление, отображенное на экране блокировки

# 15.9. Реагирование на пуш-уведомления

## Постановка задачи

Проработав раздел 15.8, вы научились доставлять в ваше приложение пуш-уведомления, но не знаете, как реагировать на них в программе.

## Решение

Реализуйте метод `application:didReceiveRemoteNotification:` делегата вашего приложения.

## Обсуждение

Метод `application:didReceiveRemoteNotification:` делегата вашего приложения вызывается всякий раз, когда в систему iOS поступает пуш-уведомление и пользова-

тель реагирует на это уведомление каким-то образом, инициируя открытие приложения. Этот метод срабатывает, когда приложение функционирует в приоритетном или в фоновом режиме, а не завершено. Например, пользователь может проигнорировать поступившее уведомление. Тогда и этот метод вызван не будет. Если пользователь нажимает на экране окно с пуш-уведомлением, в результате чего ваше приложение открывается, то iOS, открыв это приложение, переводит программу в приоритетный режим. После этого в делегате вашего приложения будет вызван вышеупомянутый метод.

Если приложение завершено и не работает даже в фоновом режиме, то iOS инкапсулирует пуш-уведомление, инициирующее запуск приложения, в параметрах запуска. Эти параметры будут переданы методу `application:didFinishLaunchingWithOptions:` делегата вашего приложения. Чтобы получить объект уведомления, просто запросите параметр `didFinishLaunchingWithOptions` этого метода (относящийся к типу `NSDictionary`) и поищите ключ `UIApplicationLaunchOptionsRemoteNotificationKey`. Значением этого ключа и будет то пуш-уведомление, которое запустило ваше приложение.

Параметр `didReceiveRemoteNotification` этого свойства несет в себе словарь типа `NSDictionary`. Этот словарь будет содержать в себе корневой объект под названием `aps`. Ниже этого объекта располагается словарь со следующими ключами, зависящими от того, как сервер создал пуш-уведомление (сервер может и не прислать все ключи сразу):

- `badge` — значением этого ключа является номер, который будет задан для ярлыка (значка) вашего приложения;
- `alert` — это содержащееся в пуш-уведомлении сообщение типа `String`. Сервер может прислать вам модифицированную версию значения этого ключа. Такое значение само по себе будет словарем, содержащим ключи `body` и `show-view`. Если вам будет прислана такая модифицированная версия предупреждения, то ключ `body` будет содержать именно тот текст, который находится в теле предупреждения. Ключ `show-view` будет содержать логическое значение, определяющее, должно ли предупреждение отображаться для пользователя. Кнопка Action (Действие) позволяет пользователю нажать на поступившее уведомление в центре уведомлений, чтобы открыть ваше приложение;
- `sound` — это строка, указывающая имя звукового файла, который должно воспроизводить ваше приложение;
- `content-available` — значением этого ключа является число. Если здесь стоит число `1`, это означает, что на сервере появился новый контент, который приложение может скачать. Сервер может послать этот ключ приложению, затребовав таким образом, чтобы оно забрало с сервера список новых элементов. Ваше приложение может и не выполнять это требование. Это просто протокол, действующий между сервером и клиентом. Если в вашем приложении это целесообразно, следуйте данному протоколу.

## См. также

Раздел 15.8.

# 16 Фреймворк Core Data

## 16.0. Введение

Core Data — это мощный фреймворк, входящий в состав iOS SDK. Он позволяет программисту сохранять данные и управлять ими объектно-ориентированным способом. Традиционно программисту приходилось сохранять данные на диске, пользуясь архивационными возможностями Objective-C, либо записывать данные в файлы, а потом управлять ими вручную. С появлением Core Data программист может просто взаимодействовать с его объектно-ориентированным интерфейсом и эффективно управлять своими данными. В этой главе будет рассмотрено, как использовать Core Data для создания модели своего приложения (с применением программной архитектуры «модель — вид — контроллер»).

Фреймворк Core Data обеспечивает низкоуровневые взаимодействия с хранилищем данных устройства, то есть взаимодействия, незаметные для программиста. iOS сама определяет, как будет организовано низкоуровневое управление данными. Для реализации такого взаимодействия программисту достаточно знать, какой высокоуровневый API для этого предназначен. Но при этом важно понимать и структуру фреймворка Core Data, его внутреннее функционирование. Чтобы лучше с этим разобраться, создадим приложение, использующее Core Data.

Теперь у нас есть новый компилятор LLVM, поэтому, чтобы включить фреймворк Core Data в ваш проект, достаточно включить обобщающий заголовок этого фреймворка, вот так:

```
#import "AppDelegate.h"
#import <CoreData/CoreData.h>

@implementation AppDelegate

<# Остаток вашего кода находится здесь #>
```

Для работы с Core Data необходимо понимать стек этого фреймворка, который составляют следующие основные элементы:

- *постоянное хранилище данных* — объект, представляющий находящуюся на диске базу данных. Мы никогда не используем этот объект непосредственно;
- *координатор постоянного хранилища данных* — объект, координирующий считывание информации из постоянного хранилища и запись в него. Координатор — это промежуточное звено между контекстом управляемых объектов и постоянным хранилищем данных;
- *модель управляемого объекта (MOM)* — обычный файл на диске, который будет представлять нашу модель данных. Считайте, что это схема базы данных;
- *управляемый объект* — этот класс представляет сущность, которую мы хотим сохранить в Core Data. В традиционном программировании баз данных такие сущности называются таблицами. Управляемый объект относится к типу `NSManagedObject`, экземпляры таких объектов помещаются в контекстах управляемых объектов. Они соответствуют схеме, заложенной в модели управляемого объекта, и сохраняются в постоянном хранилище данных с помощью координатора;
- *контекст управляемых объектов* — его можно сравнить с виртуальной приборной панелью. Странно звучит, да? Сейчас все будет понятно. Мы создаем объекты Core Data в памяти, задаем их свойства и манипулируем ими. Все эти операции происходят в контексте управляемого объекта. Контекст отслеживает все операции, совершаемые над управляемыми объектами, и даже позволяет нам отменять такие действия. Представьте другую метафору: ваши управляемые объекты, находящиеся в контексте, — это игрушки, а сам контекст — это стол, на который вы их положили. Игрушки можно передвигать на столе, разбирать, убирать какие-то со стола и класть на их место новые. Итак, стол — это контекст управляемых объектов. Закончив манипуляции с объектами, вы можете сохранить его состояние. Когда мы сохраняем состояние контекста управляемых объектов, информация об операции сохранения передается в постоянное хранилище данных. Это делается посредством координатора, к которому подключен контекст. На основании этой информации координатор хранилища данных будет записывать информацию в постоянное хранилище данных, а затем — на диск.

Чтобы добавить Core Data в свой проект, а затем приступить к использованию всех его классных возможностей, просто создайте проект. Затем, когда система спросит, добавлять ли к нему Core Data, установите соответствующий флажок (рис. 16.1).

После того как вы создадите проект с Core Data, у делегата вашего приложения появится ряд новых свойств:

```
NSManagedObjectContext *managedObjectContext;
NSManagedObjectModel *managedObjectModel;
NSPersistentStoreCoordinator *persistentStoreCoordinator;
```

Вы уже понимаете, что означают эти свойства, — они были описаны ранее в данной главе. Контекст — наш игровой стол, модель — схема базы данных, а координатор — объект, опосредующий сохранение контекста на диске. Все просто. Теперь переходим к изучению основных разделов этой главы.

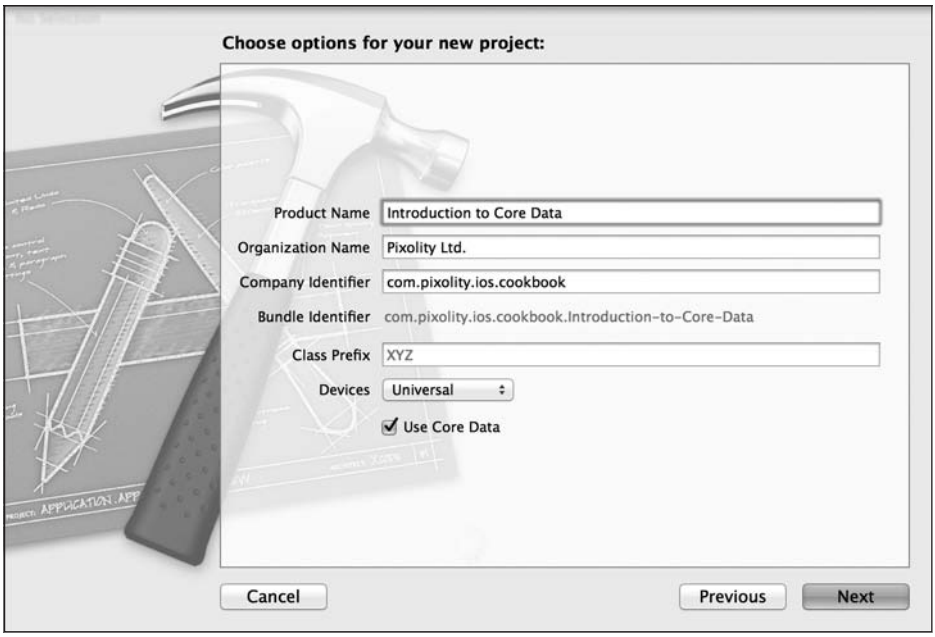


**Рис. 16.1.** Добавление Core Data к новому проекту Xcode

# 16.1. Создание модели Core Data с помощью Xcode

## Постановка задачи

Требуется визуально спроектировать в Xcode модель данных для вашего приложения iOS.

## Решение

Следуя инструкциям из введения к данной главе, создайте проект Core Data. Потом найдите в пакете вашего приложения файл с расширением `xcdatamodel` и откройте его в визуальном редакторе данных (рис. 16.2).

## Обсуждение

Визуальный редактор данных Xcode — потрясающий инструмент, позволяющий программисту с легкостью проектировать модель данных для своего приложения. Прежде чем приступить к работе с этим инструментом, необходимо усвоить два очень важных определения:

- *сущность* (Entity) — аналогична таблице базы данных;
- *атрибут* (Attribute) — аналогичен столбцу в базе данных.

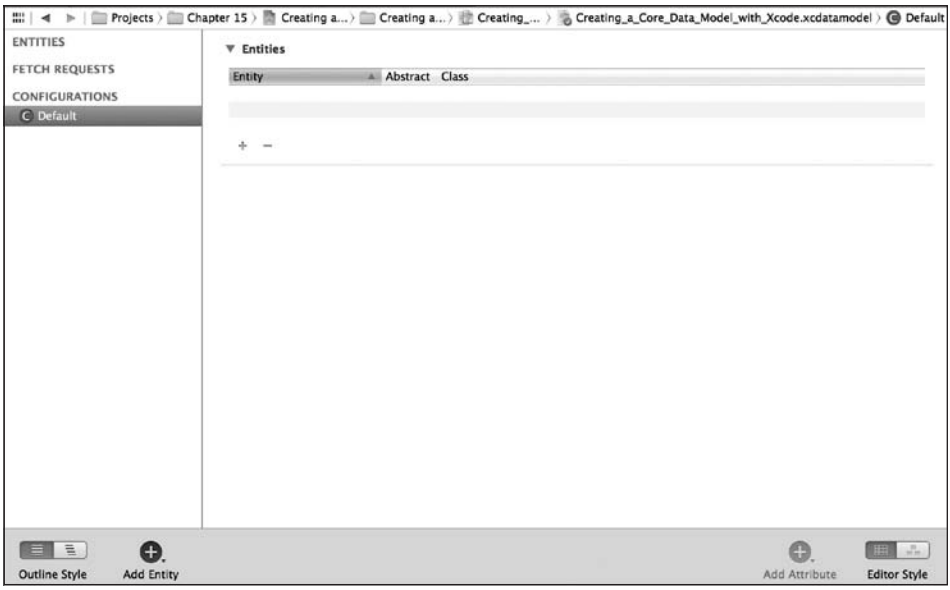


**Рис. 16.2.** Визуальный редактор данных в Xcode

Позже сущности станут объектами (управляемыми объектами). Это произойдет после того, как мы сгенерируем код на базе нашей объектной модели. Об этом пойдет речь в разделе 16.2. В текущем разделе мы сосредоточимся на создании модели данных в визуальном редакторе.

В нижней части окна редактора найдите кнопку +. Щелкните правой кнопкой мыши, удерживая указатель на этом плюсике, а потом выберите из контекстного меню вариант Add Entity (Добавить сущность) (рис. 16.3).

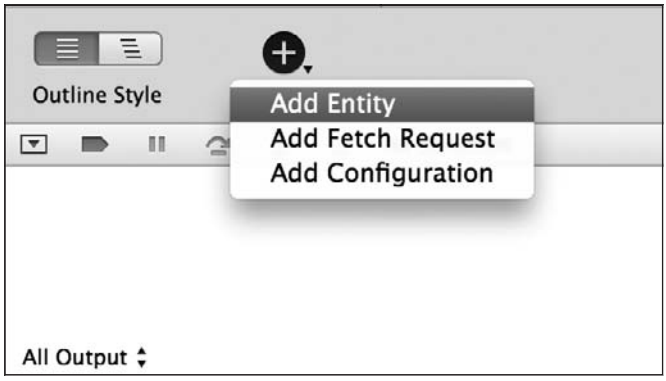


**Рис. 16.3.** Добавление новой сущности к модели данных

Сущность, которую вы создали, сразу же после создания будет находиться в состоянии, позволяющем немедленно ее переименовать. Измените название этой сущности на `Person` (Контакт) (рис. 16.4).

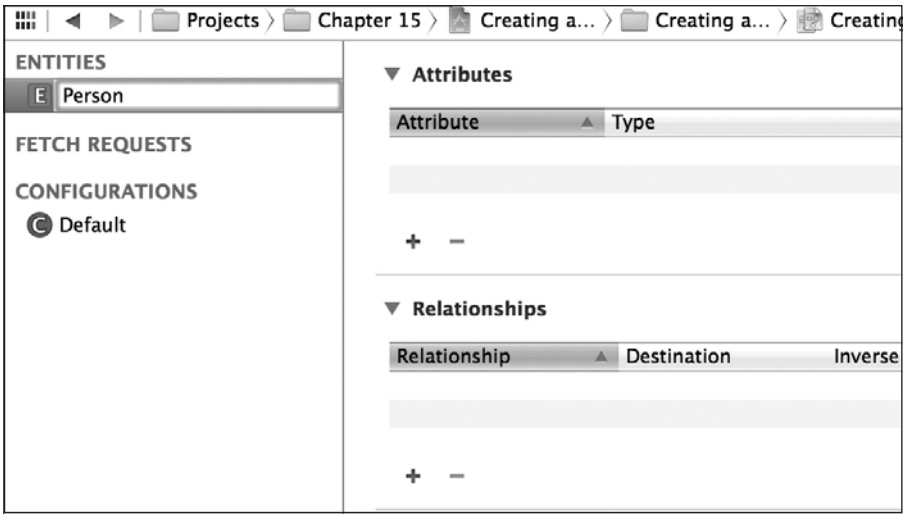


**Рис. 16.4.** Изменение имени новой сущности на Person

Выберите сущность `Person`, потом щелкните на + в области Attributes (Атрибуты) и создайте для сущности три следующих атрибута (рис. 16.5):

- `firstName` (типа `String`);
- `lastName` (типа `String`);
- `age` (типа `Integer 32`).

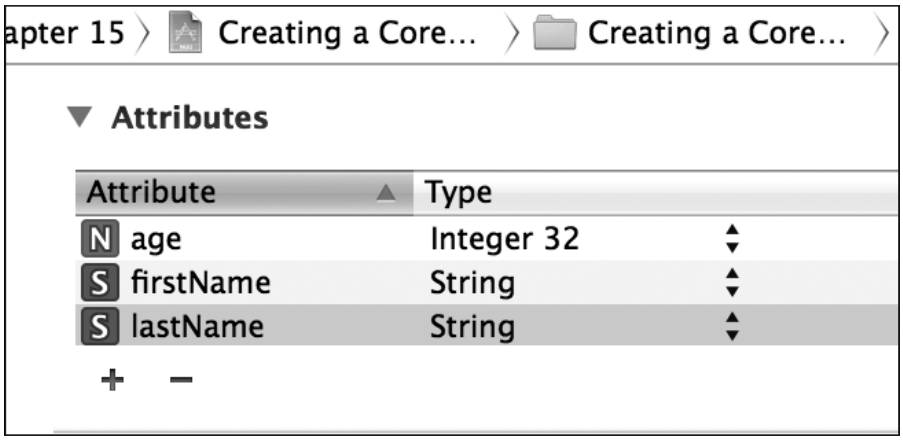


**Рис. 16.5.** Мы добавили три атрибута к сущности Person

В редакторе модели данных выберите из меню View (Вид) в Xcode команду Utilities ▸ Show Utilities (Вспомогательная область ▸ Отобразить вспомогательные возможности). В правой части Xcode откроется вспомогательная область. В верхней части этой области нажмите кнопку Data Model Inspector (Инспектор модели данных) и убедитесь, что не забыли щелкнуть на только что созданной нами сущ-

ности Person (Контакт). На данном этапе инспектор модели данных заполнится элементами, относящимися к сущности Person (рис. 16.6).

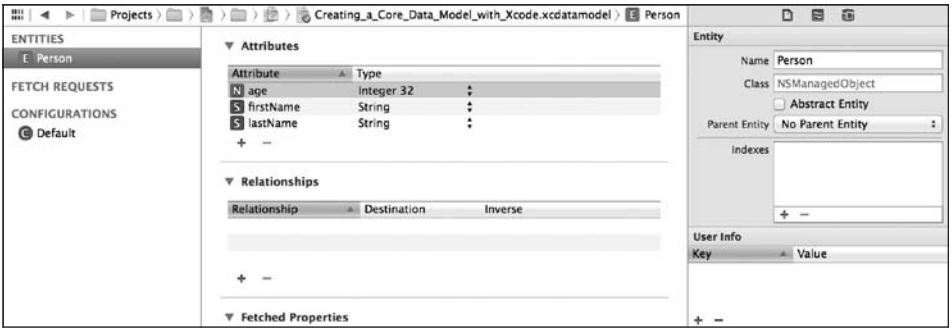


**Рис. 16.6.** Инспектор модели данных отображается в правой части окна Xcode

Теперь щелкните на атрибутах firstName, lastName и age сущности Person. Убедитесь, что атрибуты firstName и lastName *не являются опциональными* — флажок Optional должен быть снят. При этом для атрибута age флажок Optional должен быть установлен.

Итак, мы создали модель. Выполните команду File ▸ Save (Файл ▸ Сохранить), чтобы убедиться, что сделанные изменения сохранены. О том, как сгенерировать код на базе только что созданной вами модели, рассказывается в разделе 16.2.

# 16.2. Генерирование файлов классов для сущностей Core Data

## Постановка задачи

Вы выполнили все инструкции из раздела 16.1. Теперь требуется научиться создавать код на основании имеющейся объектной модели.

## Решение

Выполните следующие шаги.

1. В Xcode найдите созданный для вашего приложения файл с расширением xcdatamodel. Он был заготовлен на этапе создания самого приложения в Xcode. Щелкните на этом файле — и вы должны увидеть, как в правой части окна Xcode открывается редактор.
2. Выберите сущность Person, созданную нами ранее (см. раздел 16.1).
3. Выполните в Xcode команду File ▸ New File (Файл ▸ Новый файл).
4. В диалоговом окне New File (Новый файл) убедитесь, что выбрали iOS в качестве основной категории, а Core Data — в качестве подкатегории. Потом укажите

в правой части окна элемент NSManagedObject subclass (Подкласс NSManagedObject) и нажмите Next (Далее) (рис. 16.7).


**Рис. 16.7.** Создание в Xcode подкласса управляемого объекта

5. На следующем экране выберите модель управляемого объекта, которую вы хотите сохранить на диске, и отметьте ее флажком. Сделав это, нажмите кнопку Next (Далее) (рис. 16.8).

Если в вашем проекте всего одна модель, то и в списке вы увидите всего одну модель управляемого объекта. Но на рис. 16.8 мы видим много моделей. Дело в том, что в моем рабочем пространстве в Xcode присутствует множество проектов и каждый из них имеет свою модель.

6. Теперь система предложит вам выбрать сущности, которые вы хотите экспортировать из своей модели на диск в виде файлов Objective-C. Поскольку мы создали всего одну сущность — `Person`, ваш список будет выглядеть примерно как на рис. 16.9. Убедитесь, что сущность `Person` отмечена, а потом нажмите кнопку Next (Далее).
7. На последнем этапе вам будет предложено сохранить сущность на диске. Убедитесь, что ваш проект отмечен в поле Targets (Цели) (рис. 16.10). В противном случае эта сущность не будет доступна в других файлах исходного кода, используемых в проекте. Если вас все устраивает, нажмите кнопку Create (Создать).

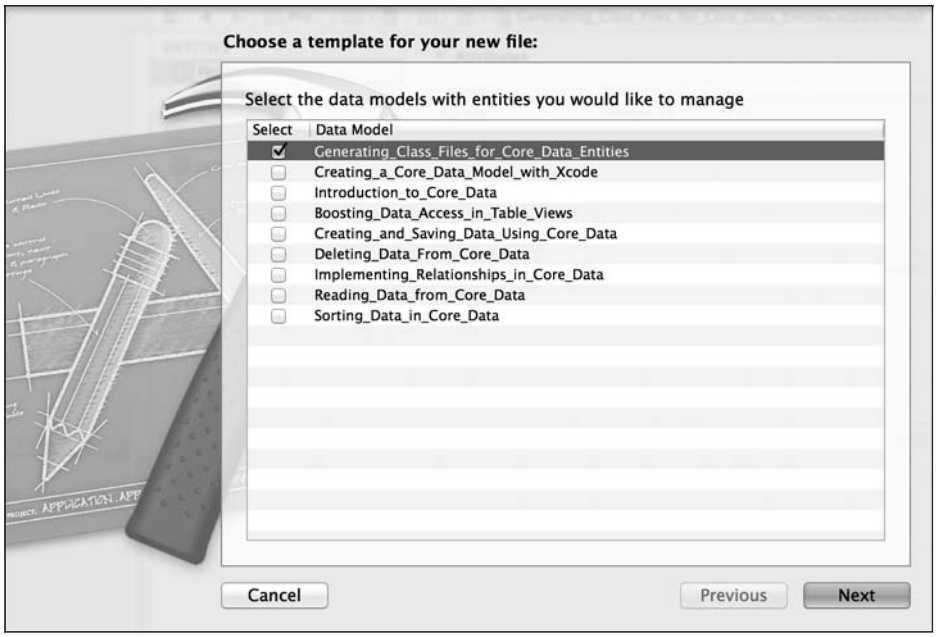


**Рис. 16.8.** Выбор модели управляемого объекта для сохранения на диске

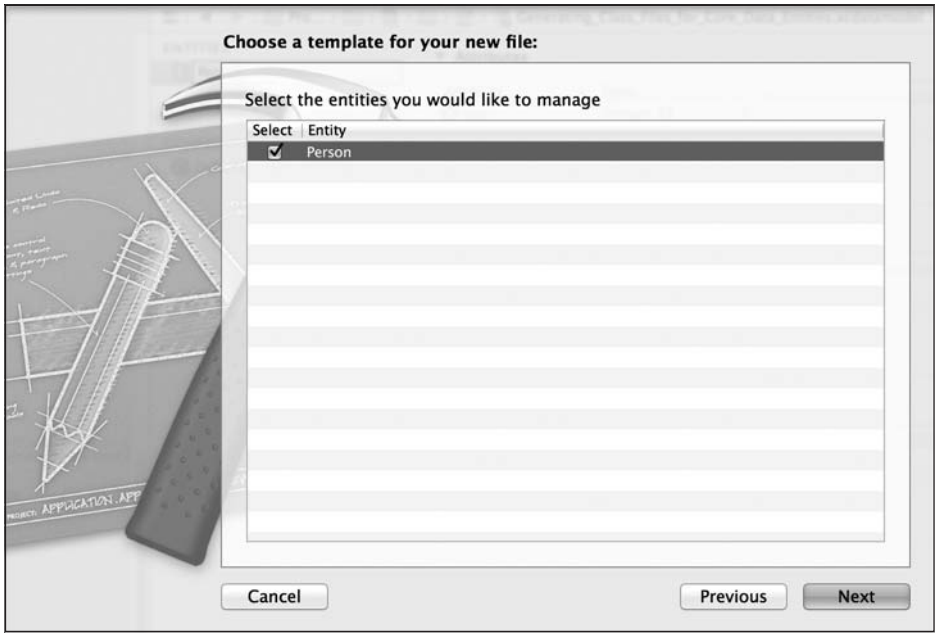


**Рис. 16.9.** Экспорт сущности Person на диск в качестве управляемого объекта

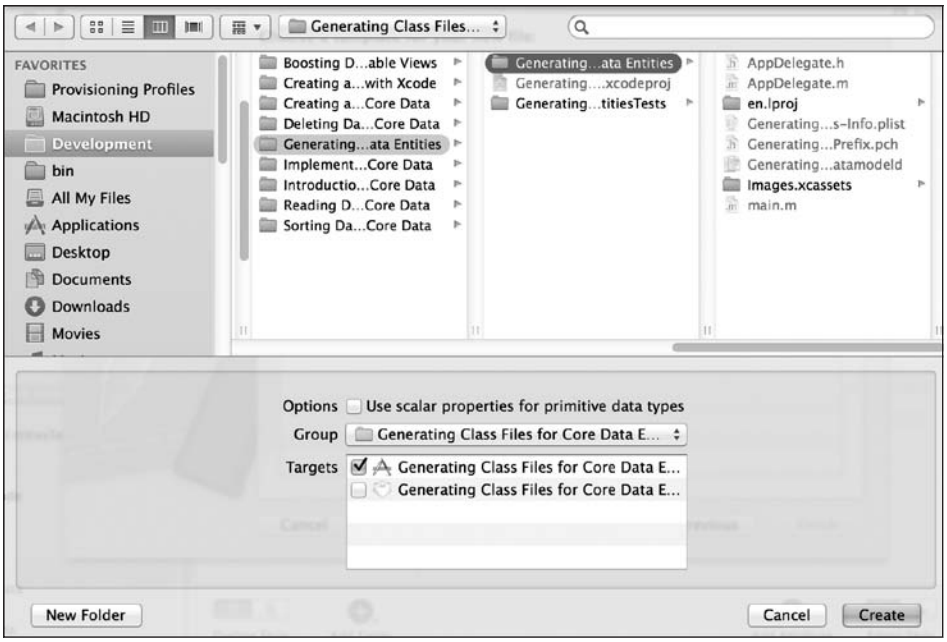


**Рис. 16.10.** Сохранение сущности на диске

Итак, вы увидите в своем проекте два файла, которые называются `Person.h` и `Person.m`. Откройте файл `Person.h`. Там будет написано следующее:

```
#import <Foundation/Foundation.h>
#import <CoreData/CoreData.h>

@interface Person : NSManagedObject
@property (nonatomic, retain) NSNumber * age;
@property (nonatomic, retain) NSString * firstName;
@property (nonatomic, retain) NSString * lastName;

@end
```

Файл `Person.m` реализуется следующим образом:

```
#import "Person.h"

@implementation Person
@dynamic age;
@dynamic firstName;
@dynamic lastName;
@end
```

Вот и все! Мы выполнили реальное определение и реализацию управляемого объекта. В разделе 16.3 мы научимся инстанцировать и сохранять управляемый объект типа `Person` в контексте управляемых объектов приложения.

## Обсуждение

Когда мы создавали в Xcode модель данных с помощью редактора, в ходе этой работы создавали отношения данных, сущности, атрибуты и т. д. Тем не менее, чтобы эту модель можно было использовать в приложении, для нее нужно сгенерировать код. Если просмотреть файлы `.h` и `.m` ваших сущностей, то выяснится, что все атрибуты присваиваются динамически. В `.m`-файле сущностей вы увидите директиву `@dynamic`. Она сообщает компилятору, что вы выполните запрос каждого атрибута во время исполнения с применением динамического расширения метода.

Код, применяемый во фреймворке Core Data к вашим сущностям, остается совершенно невидимым. На самом деле действительно нет никакой необходимости в том, чтобы программист видел этот код. Все, о чем следует знать, — сущность `Person` имеет три атрибута: `firstName`, `lastName` и `age`. Этим атрибутам можно присваивать значения (если они являются свойствами, доступными для чтения и записи), их можно сохранять в контекст и загружать из контекста, как будет показано в разделе 16.3.

# 16.3. Создание и сохранение данных с помощью Core Data

## Постановка задачи

Вы создали управляемый объект. После этого вы хотите инстанцировать его и вставить этот экземпляр в контекст Core Data вашего приложения.

## Решение

Выполните инструкции, описанные в разделах 16.1 и 16.2. Теперь можно использовать метод класса `insertNewObjectForEntityForName:inManagedObjectContext:`, относящийся к классу `NSEntityDescription`, чтобы создать новый объект типа, указанного в первом параметре этого метода. Как только будет создана новая сущность (управляемый объект), ее можно будет изменить, модифицируя ее свойства. После того как все будет готово, сохраните контекст управляемого объекта с помощью метода экземпляра `save:`, относящегося к контексту управляемых объектов.

Предполагается, что вы уже создали в Xcode универсальное приложение под названием `Creating and Saving Data Using Core Data`. Теперь, чтобы вставить новый управляемый объект в контекст, выполните следующие шаги.

1. Найдите файл под названием `Creating_and_Saving_Data_Using_Core_DataAppDelegate.m`.
2. Импортируйте файл `Person.h` в файл реализации делегата приложения:

Сущность Person мы создали в разделе 16.1.

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

@implementation AppDelegate

@synthesize managedObjectContext = _managedObjectContext;
@synthesize managedObjectModel = _managedObjectModel;
@synthesize persistentStoreCoordinator = _persistentStoreCoordinator;

<# Остаток кода вашего приложения находится здесь #>
```

В методе `application:didFinishLaunchingWithOptions:` совместно используемого делегата приложения напишем следующий код:

```
- (BOOL)            application:(UIApplication *)application
  didFinishLaunchingWithOptions:(NSDictionary *)launchOptions{

  Person *newPerson = [NSEntityDescription
                       insertNewObjectForEntityForName:@"Person"
                       inManagedObjectContext:self.managedObjectContext];

  if (newPerson != nil){

    newPerson.firstName = @"Anthony";
    newPerson.lastName = @"Robbins";
    newPerson.age = @51;

    NSError *savingError = nil;

    if ([self.managedObjectContext save:&savingError]){
      NSLog(@"Successfully saved the context.");
    } else {
      NSLog(@"Failed to save the context. Error = %@", savingError);
    }

  } else {
    NSLog(@"Failed to create the new person.");
  }
  self.window = [[UIWindow alloc] initWithFrame:
                 [[UIScreen mainScreen] bounds]];
  self.window.backgroundColor = [UIColor whiteColor];
  [self.window makeKeyAndVisible];
  return YES;
}
```

## Обсуждение

В предыдущих разделах было показано, как с помощью редактора Xcode создавать сущности и генерировать на их основе код. Далее нужно приступить к использованию этих сущностей и инстанцировать их. Для этого мы используем класс `NSEntityDescription` и вызываем метод `insertNewObjectForEntityForName:inManagedObject-`

Context: этого класса. В таком случае будет производиться поиск заданной сущности (указанной с именем NSString) в обозначенном контексте управляемых объектов. Ситуация напоминает процесс создания новой строки (управляемый объект) в таблице (сущность) базы данных (контекст управляемых объектов).

При попытке вставить в контекст управляемых объектов неизвестную сущность возникнет исключение типа NSInternalInconsistencyException.

После того как в контекст будет вставлена новая сущность, его необходимо сохранить. В результате все несохраненные данные контекста будут сброшены в долговременную память. Это можно сделать с помощью метода экземпляра save:, относящегося к нашему контексту управляемых объектов. Если логическое (BOOL) возвращаемое значение этого метода равно YES, мы можем быть уверены, что контекст сохранен. В разделе 16.4 будет рассмотрено, как считывать данные назад в оперативную память.

# 16.4. Считывание данных из Core Data

## Постановка задачи

Требуется считывать содержимое ваших сущностей (таблиц) с помощью Core Data.

## Решение

Воспользуйтесь экземпляром класса NSFetchRequest:

```
- (BOOL) createNewPersonWithFirstName:(NSString *)paramFirstName
                             lastName:(NSString *)paramLastName
                                  age:(NSUInteger)paramAge{

  BOOL result = NO;

  if ([paramFirstName length] == 0 ||
      [paramLastName length] == 0){
    NSLog(@"First and Last names are mandatory.");
    return NO;
  }

  Person *newPerson = [NSEntityDescription
                       insertNewObjectForEntityForName:@"Person"
                       inManagedObjectContext:self.managedObjectContext];

  if (newPerson == nil){
    NSLog(@"Failed to create the new person.");
    return NO;
```

```
    }

    newPerson.firstName = paramFirstName;
    newPerson.lastName = paramLastName;
    newPerson.age = @(paramAge);

    NSError *savingError = nil;

    if ([self.managedObjectContext save:&savingError]){
      return YES;
    } else {
      NSLog(@"Failed to save the new person. Error = %@", savingError);
    }

    return result;

}

- (BOOL)            application:(UIApplication *)application
    didFinishLaunchingWithOptions:(NSDictionary *)launchOptions{

    [self createNewPersonWithFirstName:@"Anthony"
                              lastName:@"Robbins"
                                   age:51];

    [self createNewPersonWithFirstName:@"Richard"
                              lastName:@"Branson"
                                   age:61];

    /* Сообщаем запросу, что мы собираемся считать содержимое
    сущности Person */
    /* Сначала создаем запрос выборки данных. */

    NSFetchRequest *fetchRequest = [[NSFetchRequest alloc]
                                    initWithEntityName:@"Person"];

    NSError *requestError = nil;
    /* И применяем к контексту запрос выборки данных. */
    NSArray *persons =
    [self.managedObjectContext executeFetchRequest:fetchRequest
                                             error:&requestError];

    /* Убеждаемся, что получили массив. */
    if ([persons count] > 0){

      /* По порядку перебираем все контакты, содержащиеся в массиве. */
      NSUInteger counter = 1;
      for (Person *thisPerson in persons){

        NSLog(@"Person %lu First Name = %@",
              (unsigned long)counter,
```

```
            thisPerson.firstName);

      NSLog(@"Person %lu Last Name = %@",
            (unsigned long)counter,
            thisPerson.lastName);

      NSLog(@"Person %lu Age = %ld",
            (unsigned long)counter,
            (unsigned long)[thisPerson.age unsignedIntegerValue]);

      counter++;
    }

  } else {
    NSLog(@"Could not find any Person entities in the context.");
  }

  self.window = [[UIWindow alloc] initWithFrame:
                 [[UIScreen mainScreen] bounds]];

  self.window.backgroundColor = [UIColor whiteColor];
  [self.window makeKeyAndVisible];
  return YES;
}
```

В данном коде мы используем переменную счетчика внутри блока быстрого перебора. Причина, по которой в ходе этого быстрого перебора требуется счетчик, заключается в том, что на консоль выводятся отладочные сообщения NSLog, которые мы просматриваем, чтобы узнать для перечисляемого в данный момент объекта его индекс в массиве. Альтернативным вариантом решения было бы использование классического for-цикла с переменной счетчика.

Подробнее о запросах выборки данных поговорим в подразделе «Обсуждение» данного раздела.

## Обсуждение

Тем, кто знаком с терминологией баз данных, *запрос выборки данных* напомнит оператор `SELECT`. В операторе `SELECT` мы указываем, какие строки и при каких условиях должны быть возвращены из какой таблицы. Запрос выборки данных делает то же самое. Мы указываем сущность (таблицу) и контекст управляемых объектов (уровень базы данных). Кроме того, можем задавать дескрипторы сортировки для данных, которые мы считываем. Но сначала поговорим о том, как упростить сам процесс считывания данных.

Чтобы считать сущность `Person` (эту сущность мы создали в разделе 16.1 и превратили ее в код в разделе 16.2), мы сначала приказываем классу `NSEntityDescription` произвести в нашем контексте управляемых объектов поиск сущности под названием `Person`. Как только она будет найдена, сообщим запросу выборки данных, что

требуется считать информацию из этой сущности. После этого нам останется всего лишь выполнить запрос выборки данных, как было показано в подразделе «Решение» данного раздела.

Метод экземпляра `executeFetchRequest:error:`, относящийся к классу `NSManagedObjectContext`, может иметь в качестве возвращаемого значения либо `nil` (в случае ошибки), либо массив управляемых объектов `Person`. Если по заданной сущности не найдено никаких результатов, то возвращенный массив будет пустым.

## См. также

Разделы 16.1 и 16.2.

# 16.5. Удаление данных из Core Data

## Постановка задачи

Требуется удалить управляемый объект (строку таблицы) из контекста управляемых объектов (вашей базы данных).

## Решение

Воспользуйтесь методом экземпляра `deleteObject:`, относящимся к классу `NSManagedObjectContext`:

```
- (BOOL)            application:(UIApplication *)application
  didFinishLaunchingWithOptions:(NSDictionary *)launchOptions{

  [self createNewPersonWithFirstName:@"Anthony"
                            lastName:@"Robbins"
                                 age:51];

  [self createNewPersonWithFirstName:@"Richard"
                            lastName:@"Branson"
                                 age:61];

  /* Сначала создаем запрос выборки данных. */
  NSFetchRequest *fetchRequest = [[NSFetchRequest alloc] init];

  NSError *requestError = nil;

  /* Теперь применим запрос выборки данных к контексту. */
  NSArray *persons =
  [self.managedObjectContext executeFetchRequest:fetchRequest
                                           error:&requestError];

/* Убеждаемся, что получили массив. */
```

```
if ([persons count] > 0){

  /* Удаляем последний контакт из массива. */
    Person *lastPerson = [persons lastObject];

    [self.managedObjectContext deleteObject:lastPerson];

    NSError *savingError = nil;
    if ([self.managedObjectContext save:&savingError]){
      NSLog(@"Successfully deleted the last person in the array.");
    } else {
      NSLog(@"Failed to delete the last person in the array.");
    }

  } else {
    NSLog(@"Could not find any Person entities in the context.");
  }

  self.window = [[UIWindow alloc] initWithFrame:
                 [[UIScreen mainScreen] bounds]];

  self.window.backgroundColor = [UIColor whiteColor];
  [self.window makeKeyAndVisible];
  return YES;
}
```

В приведенном примере кода используется метод createNewPersonWithFirstName:lastName:age:, который мы написали в разделе 16.4.

## Обсуждение

Можно удалять управляемые объекты (записи из таблицы базы данных) с помощью метода экземпляра `deleteObject:`, относящегося к классу `NSManagedObjectContext`.

Ни в одном из своих параметров этот метод не сообщает вам об ошибке, равно как и не возвращает значения `BOOL`. Таким образом, у вас нет надежного способа узнать, был ли объект успешно удален с помощью контекста управляемых объектов. Для получения этой информации лучше использовать метод `isDeleted` управляемого объекта.

С учетом данной информации изменим код, написанный ранее в этом разделе:

```
- (BOOL)            application:(UIApplication *)application
  didFinishLaunchingWithOptions:(NSDictionary *)launchOptions{

  [self createNewPersonWithFirstName:@"Anthony"
                            lastName:@"Robbins"
                                 age:51];

  [self createNewPersonWithFirstName:@"Richard"
```

```
                                lastName:@"Branson"
                                     age:61];

  /* Сначала создаем запрос выборки данных. */
  NSFetchRequest *fetchRequest = [[NSFetchRequest alloc] init];

  NSError *requestError = nil;

  /* Теперь применим запрос выборки данных к контексту. */
  NSArray *persons =
  [self.managedObjectContext executeFetchRequest:fetchRequest
                                           error:&requestError];

  /* Убеждаемся, что получили массив. */
  if ([persons count] > 0){

    /* Удаляем последний контакт из массива. */
    Person *lastPerson = [persons lastObject];

    [self.managedObjectContext deleteObject:lastPerson];

    if ([lastPerson isDeleted]){
      NSLog(@"Successfully deleted the last person...");

      NSError *savingError = nil;
      if ([self.managedObjectContext save:&savingError]){
        NSLog(@"Successfully saved the context.");
      } else {
        NSLog(@"Failed to save the context.");
      }

    } else {
      NSLog(@"Failed to delete the last person.");
    }

  } else {
    NSLog(@"Could not find any Person entities in the context.");
  }

  self.window = [[UIWindow alloc] initWithFrame:
                 [[UIScreen mainScreen] bounds]];

  self.window.backgroundColor = [UIColor whiteColor];
  [self.window makeKeyAndVisible];
  return YES;
}
```

После запуска приложения в окне консоли отобразится примерно следующий результат:

```
Successfully deleted the last person... // последний контакт успешно удален
Successfully saved the context.         // контекст успешно сохранен
```

# 16.6. Сортировка данных в Core Data

## Постановка задачи

Требуется сортировать управляемые объекты (записи), выбираемые из контекста управляемых объектов (базы данных).

## Решение

Нужно создать по экземпляру класса `NSSortDescriptor` для каждого атрибута (в терминологии баз данных — столбца) той сущности, в которой требуется произвести сортировку. Дескрипторы сортировки добавляются к массиву, а сам массив присваивается экземпляру `NSFetchRequest` с помощью метода экземпляра `setSortDescriptors:`. В данном коде, приведенном в качестве примера, `Sorting_Data_in_Core_DataAppDelegate` — это класс, представляющий делегат универсального приложения (о том, как создается сущность `Person`, рассказано в разделах 16.1 и 16.2:

```
- (BOOL)            application:(UIApplication *)application
  didFinishLaunchingWithOptions:(NSDictionary *)launchOptions{

  [self createNewPersonWithFirstName:@"Richard"
                            lastName:@"Branson"
                                 age:61];

  [self createNewPersonWithFirstName:@"Anthony"
                            lastName:@"Robbins"
                                 age:51];

  /* Сначала создаем запрос выборки данных. */
  NSFetchRequest *fetchRequest = [[NSFetchRequest alloc]
                                  initWithEntityName:@"Person"];

  NSSortDescriptor *ageSort =
  [[NSSortDescriptor alloc] initWithKey:@"age"
                              ascending:YES];

  NSSortDescriptor *firstNameSort =
  [[NSSortDescriptor alloc] initWithKey:@"firstName"
                              ascending:YES];
  fetchRequest.sortDescriptors = sortDescriptors;

  /* Сообщаем запросу, что сначала мы хотим
     считать содержимое сущности Person. */
```

```
  [fetchRequest setEntity:entity];

  NSError *requestError = nil;

  /* Теперь применим запрос выборки данных к контексту. */
  NSArray *persons =
  [self.managedObjectContext executeFetchRequest:fetchRequest
                                           error:&requestError];

  for (Person *person in persons){

    NSLog(@"First Name = %@", person.firstName);
    NSLog(@"Last Name = %@", person.lastName);
    NSLog(@"Age = %lu", (unsigned long)[person.age unsignedIntegerValue]);

  }

  self.window = [[UIWindow alloc] initWithFrame:
                 [[UIScreen mainScreen] bounds]];

  self.window.backgroundColor = [UIColor whiteColor];
  [self.window makeKeyAndVisible];
  return YES;
}
```

## Обсуждение

Экземпляр класса `NSFetchRequest` может нести с собой массив экземпляров `NSSortDescriptor`. Каждый дескриптор сортировки определяет атрибут (столбец) актуальной сущности, в которой необходимо произвести сортировку. Кроме того, он указывает порядок сортировки — восходящий или нисходящий. Например, сущность `Person`, которую мы создали в разделе 16.1, имеет атрибуты `firstName`, `lastName` и `age`. Если мы хотим считать все контакты в контексте управляемых объектов и отсортировать этих людей по возрасту, от самого младшего до самого старшего, то создадим экземпляр `NSSortDescriptor` с ключом `age` и зададим для него восходящий порядок (`ascending`):

```
NSSortDescriptor *ageSortDescriptor =
  [[NSSortDescriptor alloc] initWithKey:@"age"
                              ascending:YES];
```

Запросу выборки данных можно присвоить более одного дескриптора сортировки. Порядок расположения данных в массиве определяет и порядок, в котором задаются дескрипторы. Иными словами, вывод сортируется по первому дескриптору в массиве, в полученном множестве записи сортируются по второму дескриптору в массиве и т. д.

## См. также

Раздел 16.4.

# 16.7. Оптимизация доступа к данным в табличных видах

## Постановка задачи

Имеется приложение, в котором пользователь просматривает управляемые объекты в табличных видах. В этом приложении вы хотите выбирать и представлять данные более гибким и естественным образом, не управляя ими при этом вручную.

## Решение

Воспользуйтесь контроллерами для представления результатов выборки, которые являются экземплярами класса `NSFetchedResultsController`.

В этом разделе для ускорения разработки рассматриваемого приложения будут применены раскадровки. Подробнее о раскадровках рассказано в главе 6.

## Обсуждение

Контроллер для представления результатов выборки (Fetched Result Controller) функционально аналогичен табличному виду. Как и в таблице, в нем есть разделы и строки. Контроллер для представления результатов выборки может считывать управляемые объекты из соответствующего контекста, а также подразделять эти объекты на разделы и строки. Каждый раздел является группой, если задать такое условие в параметрах запроса, а каждая строка в разделе является управляемым объектом. Есть несколько важных причин, по которым может понадобиться модифицировать ваше приложение, чтобы в нем можно было применять контроллеры для представления результатов выборки. Эти причины таковы.

- После создания контроллера для представления результатов выборки в контексте управляемых объектов любое изменение данных (вставка, удаление, модификация и т. д.) немедленно отразится и в контроллере для представления результатов выборки. Например, можно создать контроллер для представления результатов выборки, чтобы считывать управляемые объекты сущности `Person`. Потом где-то в другой точке вашего приложения может понадобиться вставить в контекст новый управляемый объект `Person` (речь идет о том самом контексте, в котором был создан контроллер для представления результатов выборки). Сразу же после этого новый управляемый объект станет доступен в контроллере для представления результатов выборки. Чудеса, да и только!
- Имея контроллер для представления результатов выборки, можно более эффективно управлять кэшем. Например, можно указать контроллеру для представления результатов выборки сохранить в памяти только *N* управляемых объектов на каждый экземпляр такого контроллера.

- Контроллеры для представления результатов выборки аналогичны табличным видам в том отношении, что в них, как и в таблицах, есть разделы и строки — об этом говорилось ранее. Можно использовать контроллер для представления результатов выборки, чтобы без труда отображать табличные виды вашего приложения прямо в графическом пользовательском интерфейсе.

Рассмотрим некоторые важные свойства и методы экземпляра, относящиеся к контроллерам для представления результатов выборки (все объекты относятся к типу `NSFetchedResultsController`).

- `sections` (свойство типа `NSArray`) — контроллер для представления результатов выборки может группировать данные, используя путь к ключу. Выделенный инициализатор класса `NSFetchedResultsController` принимает данный группирующий фильтр в параметре `sectionNameKeyPath`. После этого в массиве `sections` будут содержаться все сгруппированные разделы. Каждый объект данного массива соответствует протоколу `NSFetchedResultsSectionInfo`.
- `objectAtIndexPath:` (метод экземпляра, возвращает управляемый объект) — объекты, выбираемые с помощью описываемого контроллера, их можно получать по индексу в разделе или строке. Строки каждого раздела нумеруются от 0 до $N - 1$, где $N$ — общее количество элементов в данном разделе. В объекте пути к индексу указывается как индекс раздела, так и индекс строки, и в результате этого совершенно точно формулируется информация, необходимая для получения конкретных объектов от контроллера для представления результатов выборки. Метод экземпляра `objectAtIndexPath` принимает индексные пути. Каждый индексный путь — это объект типа `NSIndexPath`. Если требуется создать ячейку табличного вида, воспользовавшись управляемым объектом из контроллера для представления результатов выборки, то нужно просто передать объект индексного пути методу делегата табличного вида `tableView:cellForRowAtIndexPath:`. Такая передача происходит в параметре `cellForRowAtIndexPath` этого метода. Если вы хотите сами создать индексный путь в любой другой точке вашего приложения, пользуйтесь методом класса `indexPathForRow:inSection:`, относящимся к классу `NSIndexPath`.
- `fetchRequest` (свойство типа `NSFetchRequest`) — если в любой точке вашего приложения возникнет необходимость заменить объект запроса выборки контроллером для представления результатов выборки, то это можно сделать с помощью свойства `fetchRequest` экземпляра `NSFetchedResultsController`. Такая возможность будет полезна, например, если необходимо изменить дескрипторы сортировки (о них подробно рассказано в разделе 16.6) запроса выборки — уже после того, как вы выделили и инициализировали ваши контроллеры для представления результатов выборки.

Контроллер для представления результатов выборки также отслеживает изменения, происходящие в том контексте, с которым он связан. Допустим, контроллер для представления результатов выборки создан в контроллере вида A, а в контроллере вида B мы удаляем объект из нашего контекста. Поскольку удаление происходит в контроллере вида B, первый контроллер вида A, владеющий контроллером для представления результатов выборки, будет об этом уведомлен. При этом пред-

полагается, что контроллер вида A является делегатом контроллера для представления результатов выборки. Такое соотношение контроллеров удобно и очень нам пригодится. Предположим следующее: мы разрабатываем приложение, в котором пользователь видит на экране два контроллера вида. Корневой контроллер вида является табличным. В нем перечислены все пользовательские контакты. Во втором контроллере вида пользователь может добавить новый контакт. Как только пользователь нажмет в контроллере вида кнопку Save (Сохранить) и вернется к списку своих контактов, этот список уже будет обновлен благодаря механизму делегирования, действующему в контроллере, представляющем результаты выборки.

В описанном приложении потребуется объявить табличный контроллер вида, в котором все пользовательские контакты перечислены следующим образом:

```
#import "PersonsListTableViewController.h"
#import "AppDelegate.h"
#import "Person.h"
#import "AddPersonViewController.h"

static NSString *PersonTableViewCell = @"PersonTableViewCell";

@interface PersonsListTableViewController ()
<NSFetchedResultsControllerDelegate>

@property (nonatomic, strong) UIBarButtonItem *barButtonAddPerson;
@property (nonatomic, strong) NSFetchedResultsController *frc;

@end
```

Кнопка панели, объявленная в этом коде, будет представлять собой простую кнопку +. Этот плюсик будет находиться на навигационной панели. Такая кнопка позволяет пользователю перейти в контроллер вида `Add Person` (Добавить контакт), где можно будет добавить новый контакт в имеющийся контекст управляемых объектов. Контроллер для представления результатов выборки также будет использоваться для выборки контактов из контекста и последующего их отображения в табличном виде.

Вот как создается контроллер для представления результатов выборки:

```
/* Сначала создаем запрос выборки данных */

NSFetchRequest *fetchRequest = [[NSFetchRequest alloc]
                                initWithEntityName:@"Person"];

NSSortDescriptor *ageSort =
[[NSSortDescriptor alloc] initWithKey:@"age"
                           ascending:YES];

NSSortDescriptor *firstNameSort =
[[NSSortDescriptor alloc] initWithKey:@"firstName"
                           ascending:YES];

fetchRequest.sortDescriptors = @[ageSort, firstNameSort];
```

```
self.frc =
[[NSFetchedResultsController alloc]
 initWithFetchRequest:fetchRequest
 managedObjectContext:[self managedObjectContext]
   sectionNameKeyPath:nil
            cacheName:nil];

self.frc.delegate = self;
NSError *fetchingError = nil;
if ([self.frc performFetch:&fetchingError]){
  NSLog(@"Successfully fetched.");
} else {
  NSLog(@"Failed to fetch.");
}
```

Как видите, контроллер для представления результатов выборки принимает контроллер актуального табличного вида в качестве своего делегата. Делегат контроллера для представления результатов выборки должен соответствовать протоколу `NSFetchedResultsControllerDelegate`. Вот некоторые из наиболее важных методов этого протокола.

- `controllerWillChangeContent:` — вызывается в делегате и сообщает ему об изменении контекста, служащего основой для контроллера, представляющего результаты выборки, а также о том, что содержимое контроллера, представляющего результаты выборки, вот-вот изменится с учетом внесенных изменений. Обычно этот метод используется для подготовки табличного вида к изменениям. Для этого в нем вызывается метод `beginUpdates`.
- `controller:didChangeObject:atIndexPath:forChangeType:newIndexPath:` — вызывается в делегате и сообщает ему о конкретных изменениях, сделанных в объекте из контекста. Например, если вы удаляете объект в контексте, то вызывается этот метод. При этом его параметр `forChangeType` содержит значение `NSFetchedResultsChangeDelete`. В другом случае, когда вы вставляете новый объект в контекст, этот параметр содержит значение `NSFetchedResultsChangeInsert`.

  Кроме того, этот метод вызывается в методе делегата контроллера для представления результатов выборки, когда обновляется управляемый объект. Это происходит после того, как объект будет сохранен в объекте с помощью метода `save:`.
- `controllerDidChangeContent:` — вызывается в делегате и информирует его о том, что контроллер для представления результатов выборки был обновлен в результате обновления контекста управляемых объектов. Как правило, именно внутри этого метода программисты совершают вызов `endUpdates`, применяемый в табличных видах для обработки всех обновлений, поступивших в таблицу после срабатывания метода `beginUpdates`.

Вот типичная реализация вышеупомянутых методов в приложении, которое было описано ранее в этом разделе:

```
- (void) controllerWillChangeContent:(NSFetchedResultsController *)controller{
  [self.tableView beginUpdates];
}

- (void) controller:(NSFetchedResultsController *)controller
    didChangeObject:(id)anObject
        atIndexPath:(NSIndexPath *)indexPath
      forChangeType:(NSFetchedResultsChangeType)type
       newIndexPath:(NSIndexPath *)newIndexPath{

    if (type == NSFetchedResultsChangeDelete){
      [self.tableView
       deleteRowsAtIndexPaths:@[indexPath]
             withRowAnimation:UITableViewRowAnimationAutomatic];
    }

    else if (type == NSFetchedResultsChangeInsert){
      [self.tableView
       insertRowsAtIndexPaths:@[newIndexPath]
             withRowAnimation:UITableViewRowAnimationAutomatic];
    }

}

- (void) controllerDidChangeContent:(NSFetchedResultsController *)controller{
  [self.tableView endUpdates];
}
```

Остановимся также на передаче информации в табличный вид с помощью различных методов контроллера для представления результатов выборки — об этом мы также упоминали ранее. Одним из таких методов является `object-AtIndexPath:`. Простая реализация этого метода в табличном виде может выглядеть примерно так:

```
- (NSInteger)tableView:(UITableView *)tableView
numberOfRowsInSection:(NSInteger)section{

  id <NSFetchedResultsSectionInfo> sectionInfo =
  self.frc.sections[section];
  return sectionInfo.numberOfObjects;
}

- (UITableViewCell *)tableView:(UITableView *)tableView
cellForRowAtIndexPath:(NSIndexPath *)indexPath{

  UITableViewCell *cell = nil;

  cell = [tableView dequeueReusableCellWithIdentifier:PersonTableViewCell
                                         forIndexPath:indexPath];

  Person *person = [self.frc objectAtIndexPath:indexPath];
```

```
  cell.textLabel.text =
  [person.firstName stringByAppendingFormat:@" %@", person.lastName];

  cell.detailTextLabel.text =
  [NSString stringWithFormat:@"Age: %lu",
   (unsigned long)[person.age unsignedIntegerValue]];

  return cell;
}
```

В этом коде мы приказываем нашему контроллеру табличного вида отобразить столько ячеек, сколько экземпляров управляемых объектов находится в контроллере для представления результатов выборки. Отображая каждую ячейку, мы получаем управляемый объект `Person` из контроллера, представляющего результаты выборки, после чего соответствующим образом конфигурируем ячейку. Контроллер табличного вида, не содержащий никаких элементов в контексте управляемых объектов, будет выглядеть примерно как на рис. 16.11.


**Рис. 16.11.** Пустой табличный вид, построенный на базе контроллера для представления результатов выборки

Переходим ко второму контроллеру вида, где пользователь может добавить новый экземпляр `Person` в контекст управляемых объектов. Воспользуемся следующим методом:

```
- (void) createNewPerson:(id)paramSender{

  AppDelegate *appDelegate = [[UIApplication sharedApplication] delegate];

  NSManagedObjectContext *managedObjectContext =
  appDelegate.managedObjectContext;

  Person *newPerson =
  [NSEntityDescription insertNewObjectForEntityForName:@"Person"
```

```
   inManagedObjectContext:managedObjectContext];

  if (newPerson != nil){

    newPerson.firstName = self.textFieldFirstName.text;
    newPerson.lastName = self.textFieldLastName.text;
    newPerson.age = @([self.textFieldAge.text integerValue]);

    NSError *savingError = nil;

    if ([managedObjectContext save:&savingError]){
      [self.navigationController popViewControllerAnimated:YES];
    } else {
      NSLog(@"Failed to save the managed object context.");
    }

  } else {
    NSLog(@"Failed to create the new person object.");
  }

}
```

Этот метод считывает имя, фамилию и возраст человека. На основе этих трех информационных фрагментов в контроллере вида будет создаваться контакт. Нам не придется заниматься реализацией этих текстовых полей, поскольку такая работа никак не связана с темой данного раздела. После вызова метода мы вызываем в контексте управляемого объекта метод `save:`. Он, в свою очередь, инициирует изменения в контроллере вида для представления результатов выборки (он находится в табличном виде). В результате всего этого табличный вид обновится.

Наконец, мы должны предоставить пользователю возможность удалять элементы в контроллере первого (табличного) вида:

```
- (void) tableView:(UITableView *)tableView
commitEditingStyle:(UITableViewCellEditingStyle)editingStyle
forRowAtIndexPath:(NSIndexPath *)indexPath{

  Person *personToDelete = [self.frc objectAtIndexPath:indexPath];

  [[self managedObjectContext] deleteObject:personToDelete];
  if ([personToDelete isDeleted]){
    NSError *savingError = nil;
    if ([[self managedObjectContext] save:&savingError]){
      NSLog(@"Successfully deleted the object");
    } else {
      NSLog(@"Failed to save the context with error = %@", savingError);
    }
  }
}
```

Этот код даже не затрагивает непосредственно сам контроллер для представления результатов выборки, но удаляет выбранный контакт из контекста управляемых

объектов. В результате обновляется содержимое контроллера, представляющего результаты выборки, а это, в свою очередь, приводит к обновлению табличного вида. Подробнее о табличных видах рассказано в главе 4. Интерфейс нашего контроллера табличного вида в режиме удаления может выглядеть, примерно как на рис. 16.12.

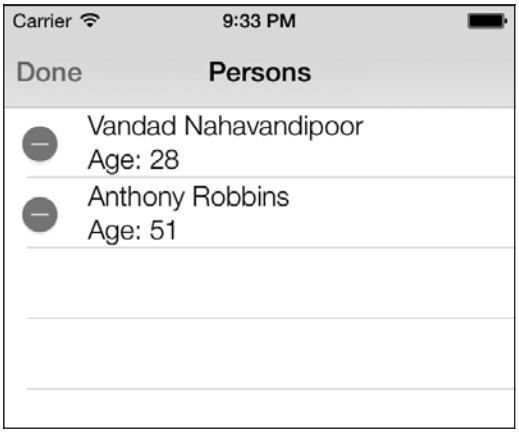


**Рис. 16.12.** Табличный контроллер вида в режиме удаления, также здесь используется контроллер вида для представления результатов выборки

# 16.8. Реализация отношений в Core Data

## Постановка задачи

Необходимо иметь возможность связывать управляемые объекты друг с другом, например связать контакт `Person` с каталогом Home, в котором он находится.

## Решение

Применяйте в редакторе модели обратные отношения.

## Обсуждение

В Core Data могут существовать следующие виды отношений: «один к одному» (one-to-one), обратное отношение «один ко многим» или обратное отношение «многие ко многим». Далее приведены жизненные примеры каждой разновидности отношений.

- *Отношение «один к одному»* — существует между человеком и его носом. У каждого человека может быть только один нос, и каждый нос может принадлежать только одному человеку.
- *Обратное отношение «один ко многим»* — существует между сотрудником и его менеджером. У сотрудника может быть только один непосредственный мене-

джер, но одному менеджеру могут одновременно подчиняться несколько сотрудников. В данном случае для сотрудника создается отношение «один к одному», однако для менеджера это отношение «один (менеджер) ко многим (сотрудникам)». Поэтому такое отношение и называется обратным.

- *Обратное отношение «многие ко многим»* — возникает между человеком и автомобилем. Одна машина может использоваться несколькими людьми, а один человек может пользоваться несколькими машинами.

В Core Data можно создавать отношения «один к одному», но я категорически не рекомендую этого делать. Возвращаясь к недавнему примеру с носом, необходимо отметить, что человек будет знать, чей нос торчит у него на лице, а вот нос не будет знать, кому он принадлежит. Обратите внимание на то, что эта система отношений «один к одному» отличается от взаимно однозначных отношений, с которыми вы могли столкнуться в других системах управления базами данных: объект A и объект B будут взаимосвязаны друг с другом, если между ними существует отношение «один к одному». В Core Data при отношении «один к одному» объект A будет знать, что связан с объектом B, но не наоборот. В объектно-ориентированном языке, таком как Objective-C, всегда лучше создавать обратные отношения, такие, которые позволяют дочерним элементам обращаться к родительским. При отношении «один ко многим» объект, который может быть ассоциирован с рядом других объектов, будет удерживать это множество объектов. Это будет множество типа `NSSet`. Хотя при отношениях «один к одному» оба объекта, состоящие в таких отношениях, сохраняют ссылку друг на друга, так как используют правильное имя класса «напарника», это отношение все равно принадлежит к типу «один к одному», и один объект может быть представлен в другом путем простого указания своего имени класса.

Итак, создадим такую модель данных, в которой используются преимущества обратного отношения «один ко многим».

1. Найдите в Xcode файл `xcdatamodel`, созданный системой в самом начале работы с проектом Core Data. Это было показано во введении к данной главе (создание такого проекта описано в разделе 16.1).
2. Откройте в редакторе файл модели данных, щелкнув на нем кнопкой мыши.
3. Удалите все созданные ранее сущности, выделяя их и нажимая клавишу Delete.
4. Создайте новую сущность и назовите ее `Employee` (Сотрудник). Создайте для этой сущности три атрибута, которые будут называться `firstName` (типа `String`), `lastName` (типа `String`) и `age` (типа `Integer 32`) (рис. 16.13).
5. Создайте сущность под названием `Manager` (Менеджер) с такими же атрибутами, как и у сущности `Employee`: `firstName` (типа `String`), `lastName` (типа `String`) и `age` (типа `Integer 32`) (рис. 16.14).
6. Создайте новое отношение для сущности `Manager`. Для этого сначала нужно выбрать данную сущность из списка, а потом нажать кнопку + в нижней части области Relationships (Отношения) (рис. 16.15).
7. В качестве имени нового отношения задайте `employees` (Сотрудники) (рис. 16.16).

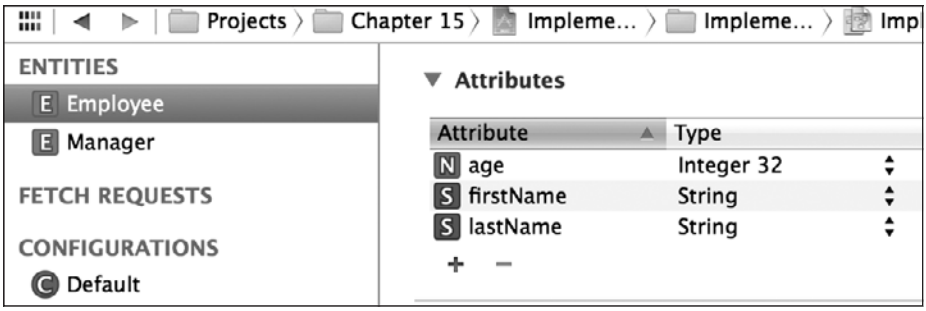


**Рис. 16.13.** Сущность Employee с тремя атрибутами

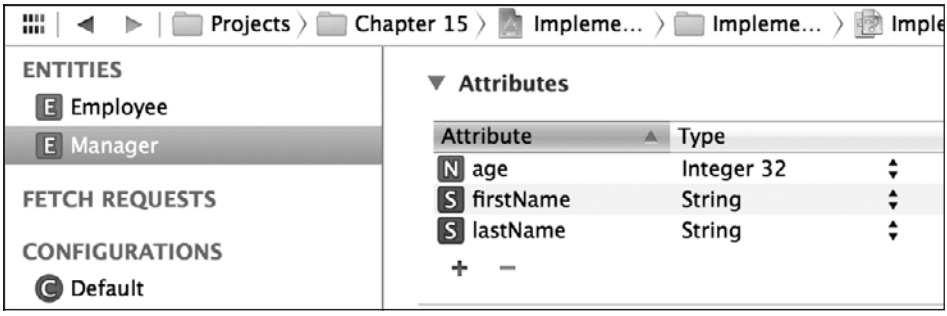


**Рис. 16.14.** Сущность Manager с тремя атрибутами

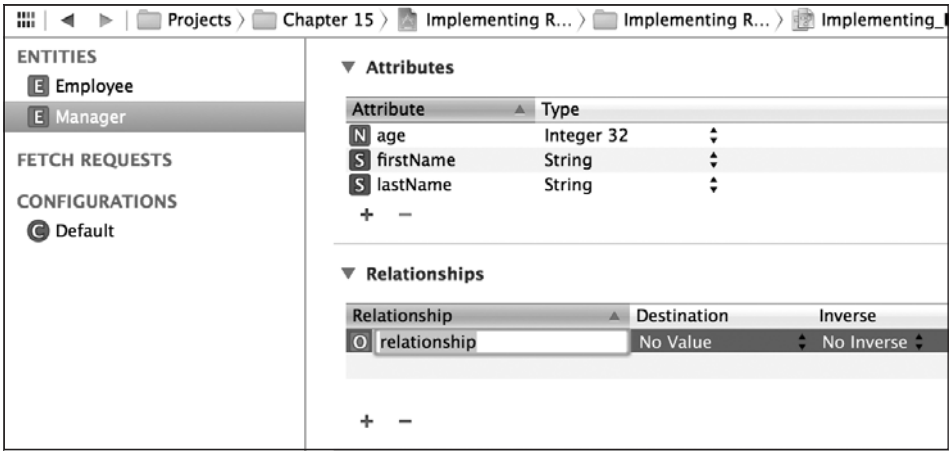


**Рис. 16.15.** Добавление нового отношения к сущности Manager

8. Выберите сущность `Employee` и создайте для нее новое отношение. Назовите это отношение `manager` (рис. 16.17).
9. Выберите сущность `Manager`, а потом выделите отношение `employees` для `Manager`. В области Relationships (Отношения) выберите параметр `Employee`

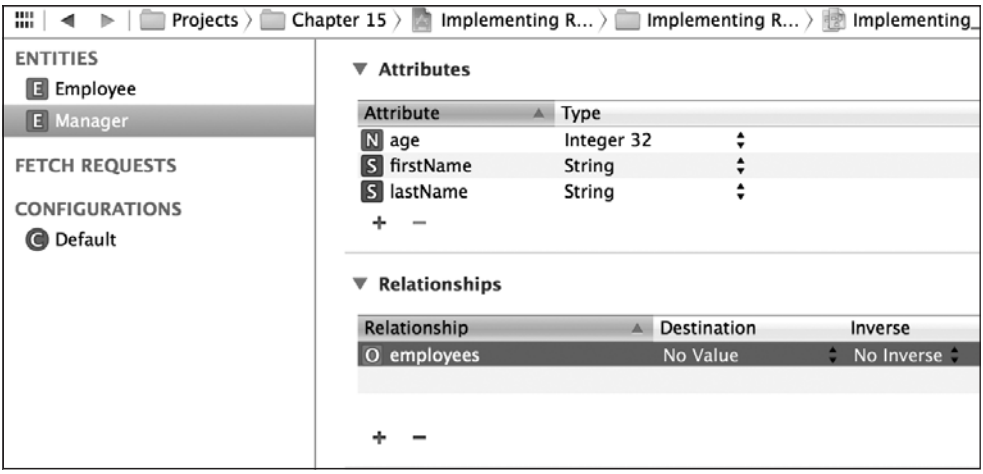


**Рис. 16.16.** Изменение имени нового отношения типа «менеджер к сотрудникам»

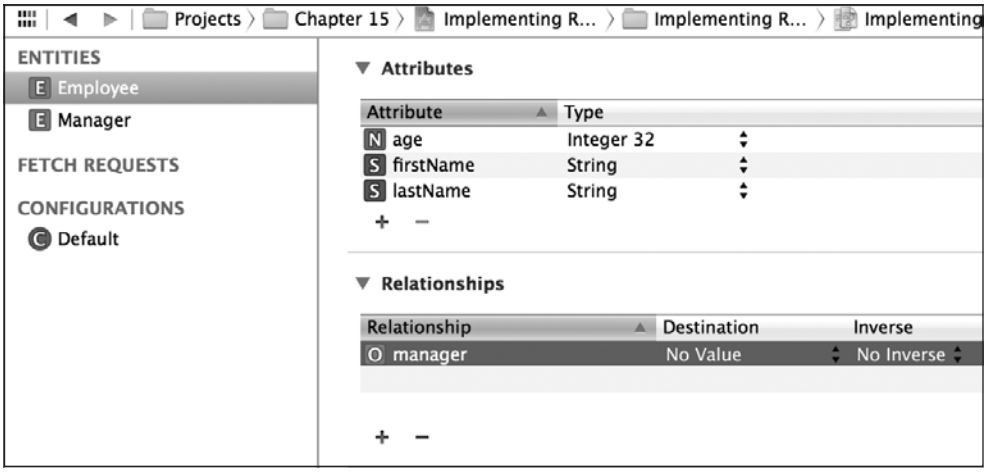


**Рис. 16.17.** Изменение имени нового отношения между сотрудниками и менеджером

(Сотрудник) в раскрывающемся меню Destination (Назначение). Именно так — ведь в этом отношении мы хотим соединить сущности `Manager` и `Employee`. В столбце Inverse (Обратные отношения) укажите значение `manager` (так как отношение `manager` будет связывать сотрудника (`Employee`) с менеджером (`Manager`)). Наконец, установите флажок To-Many Relationship (Отношение ко многим) в инспекторе модели данных (см. раздел 16.1). Результаты приведены на рис. 16.18.

10. Выделите обе сущности (`Employee` и `Manager`), выполните команду File ▸ New File (Файл ▸ Новый файл) и создайте классы управляемых объектов для вашей модели, как описано в разделе 16.2.

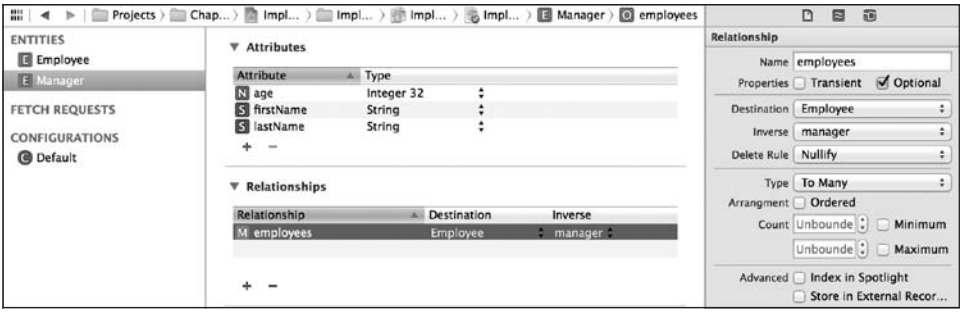


**Рис. 16.18.** Обратное отношение, установленное между менеджером и сотрудниками

Создав обратное отношение «один ко многим», откройте `.h`-файл вашей сущности `Employee`:

```
#import <Foundation/Foundation.h>
#import <CoreData/CoreData.h>

@class Manager;

@interface Employee : NSManagedObject
@property (nonatomic, retain) NSNumber * age;
@property (nonatomic, retain) NSString * firstName;
@property (nonatomic, retain) NSString * lastName;
@property (nonatomic, retain) Manager *manager;

@end
```

Как видите, в этом файле появилось новое свойство. Оно называется `manager` и относится к типу `Manager`. Таким образом, начиная с данного момента мы при наличии ссылки на конкретный объект типа `Employee` можем получить доступ к свойству `manager`, а через него — к объекту `Manager` данного конкретного сотрудника (если менеджер есть). Рассмотрим `.h`-файл сущности `Manager`:

```
#import <Foundation/Foundation.h>
#import <CoreData/CoreData.h>

@class Employee;

@interface Manager : NSManagedObject
@property (nonatomic, retain) NSNumber * age;
@property (nonatomic, retain) NSString * firstName;
@property (nonatomic, retain) NSString * lastName;
@property (nonatomic, retain) NSSet *employees;
@end

@interface Manager (CoreDataGeneratedAccessors)

- (void)addFKManagerToEmployeesObject:(Employee *)value;
- (void)removeFKManagerToEmployeesObject:(Employee *)value;
```

```
- (void)addFKManagerToEmployees:(NSSet *)values;
- (void)removeFKManagerToEmployees:(NSSet *)values;
@end
```

Для сущности `Manager` также создается свойство `employees`. Тип данных этого объекта — `NSSet`. Это означает, что свойство `employees` любого экземпляра сущности `Manager` может содержать от 1 до *N* сущностей `Employee`. В этом и заключается принцип отношения «один ко многим»: один менеджер, несколько сотрудников.

Другой тип отношений, которые, возможно, потребуется реализовать, называется «многие ко многим». По сравнению с отношением `Manager` к `Employee` при отношении «многие ко многим» один менеджер может иметь *N* сотрудников, а каждый сотрудник может подчиняться *N* менеджерам. Чтобы организовать такие отношения, выполните те же инструкции, что и при создании отношения «один ко многим», но выделите сущность `Employee`, а потом отношение `manager`. Измените это название на `managers` и установите флажок To-Many Relationship (Отношение ко многим) (рис. 16.19). Теперь стрелка будет заострена с обоих концов.


**Рис. 16.19.** Создание отношения «многие ко многим» между сущностями Manager и Employee

Теперь, открыв файл `Employee.h`, вы увидите, что его содержимое изменилось:

```
#import <Foundation/Foundation.h>
#import <CoreData/CoreData.h>

@class Manager;

@interface Employee : NSManagedObject

@property (nonatomic, retain) NSNumber * age;
@property (nonatomic, retain) NSString * firstName;
@property (nonatomic, retain) NSString * lastName;
@property (nonatomic, retain) NSSet *managers;
@end

@interface Employee (CoreDataGeneratedAccessors)

- (void)addManagersObject:(Manager *)value;
- (void)removeManagersObject:(Manager *)value;
- (void)addManagers:(NSSet *)values;
```

```
- (void)removeManagers:(NSSet *)values;

@end
```

Как видите, свойство managers сущности Person теперь представляет собой множество. Поскольку отношение сотрудника к его менеджерам — это множество и такое же отношение существует между менеджером и сотрудниками, здесь мы имеем пример отношения «многие ко многим»

В коде, написанном для отношения «один ко многим», можно просто создать новый управляемый объект Manager (о том, как вставлять объекты в контекст управляемых объектов, рассказано в разделе 16.3), сохранить его в контексте управляемых объектов, а потом соединить с парой управляемых объектов Employee — и их тоже сохранить в контексте. Теперь, чтобы ассоциировать менеджера с сотрудником, задайте в качестве значения для свойства FKEmployeeToManager, относящегося к экземпляру Employee, экземпляр управляемого объекта Manager. После этого фреймворк Core Data сам создаст необходимое отношение.

Если потребуется получить всех сотрудников (типа Employee), ассоциированных с объектом-менеджером (типа Manager), нужно будет просто воспользоваться методом экземпляра allObjects, относящимся к свойству FKManagerToEmployees вашего объекта-менеджера. Это объект типа NSSet, поэтому можно применить метод экземпляра allObjects, чтобы получить массив всех объектов-сотрудников, ассоциированных с конкретным объектом-менеджером.

# 16.9. Выборка данных в фоновом режиме

## Постановка задачи

Требуется выполнять операции выборки данных в стеке Core Data, причем только в фоновом режиме. Это отличная возможность создать по-настоящему отзывчивый пользовательский интерфейс.

## Решение

Перед тем как заниматься выборкой данных в фоновом режиме, создайте новый контекст управляемых объектов с параллелизмом типа NSPrivateQueueConcurrencyType. Затем воспользуйтесь методом performBlock: нового фонового контекста для выборки данных в фоновом режиме. Как только это будет сделано и вы будете готовы использовать выбранные объекты в пользовательском интерфейсе, вернитесь в поток пользовательского интерфейса с помощью dispatch_async (см. раздел 7.4). Далее для каждого объекта, выбранного в фоновом режиме, выполните в основном контексте метод objectWithID:. Так объекты, выбранные в фоновом режиме, будут перенесены в ваш приоритетный контекст, где вы сможете оперировать ими в потоке пользовательского интерфейса.

## Обсуждение

Выборка объектов в основном потоке — не самая хорошая идея. Выполнять ее в главном потоке можно лишь в случаях, когда в стеке Core Data совсем немного элементов. Дело в том, что при операции выборки в Core Data обычно выполняется поисковый вызов. Затем этот вызов должен выбрать для вас определенные данные, обычно с помощью предиката. Чтобы сделать пользовательский интерфейс более отзывчивым, лучше всего выполнять такие операции выборки в фоновом контексте.

Вы можете создать в приложении столько контекстов, сколько захотите, однако помните об одном железном правиле. Нельзя передавать управляемые объекты между контекстами в разных потоках, так как объекты не являются потокобезопасными. Таким образом, если вы выбираете объекты в фоновом контексте, то не можете использовать их в главном потоке. Вот как следует передавать управляемые объекты между потоками: объект выбирается в фоновом потоке, а потом переносится в главный контекст (контекст, работающий в основном потоке). Это делается с помощью метода `objectWithID:` главного контекста. Этот метод принимает объект типа `NSManagedObjectID`. Поэтому в фоновом потоке мы на самом деле не выбираем управляемые объекты как таковые, а лишь берем их сохраняемые ID, после чего передаем эти ID главному контексту, который сам получает для вас управляемый объект. Итак, вы выполняете в фоновом режиме и поиск, и выборку объектов, затем передаете ID найденных объектов главному контексту. Получением самих объектов занимается уже главный контекст. Если действовать таким образом, главный контекст будет располагать сохраняемыми ID объектов, а получение этих объектов из постоянного хранилища в такой ситуации протекает гораздо быстрее, чем при выполнении полномасштабного поиска в главном контексте.

В этом разделе предполагается, что вы уже создали модель управляемых объектов `Person`. Подобная модель показана на рис. 16.20.

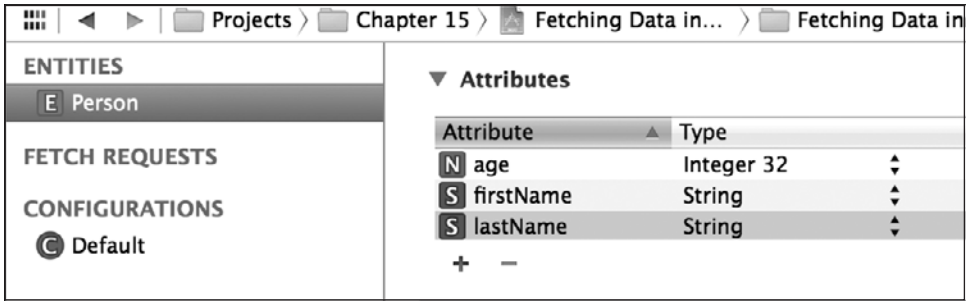


**Рис. 16.20.** Простая модель Core Data, используемая в этом разделе

При работе с этой моделью я заполню стек 1000 объектов `Person`, как показано в следующем коде, а уже потом попробую выбирать информацию из стека:

```
- (void) populateDatabase{

  for (NSUInteger counter = 0; counter < 1000; counter++){
```

```
    Person *person =
    [NSEntityDescription
     insertNewObjectForEntityForName:NSStringFromClass([Person class])
     inManagedObjectContext:self.managedObjectContext];

    person.firstName = [NSString stringWithFormat:@"First name %lu",
                        (unsigned long)counter];

    person.lastName = [NSString stringWithFormat:@"Last name %lu",
                       (unsigned long)counter];
    person.age = @(counter);
  }

  NSError *error = nil;
  if ([self.managedObjectContext save:&error]){
    NSLog(@"Managed to populate the database.");
  } else {
    NSLog(@"Failed to populate the database. Error = %@", error);
  }

}
```

Обратите внимание: я использую класс NSStringFromClass для преобразования имени класса Person в строку и для последующего инстанцирования объектов такого типа. Некоторые программисты предпочитают типизировать Person как строковый литерал. Но если жестко запрограммировать вашу строку таким образом, может возникнуть проблема. Допустим, позже вы решите изменить имя Person в стеке Core Data, а жестко закодированная строка никуда не денется. Она может привести к аварийному завершению вашего приложения во время исполнения, так как объекта модели с именем Person больше не существует. Но если вы примените вышеупомянутую функцию для преобразования имени класса в обычную строку, то при изменении имени класса или отсутствии такого класса получите ошибку времени компиляции. Такие ошибки выявляются еще до ввода приложения в работу, и у вас будет время их исправить.

Прежде чем продолжать обсуждение, оговорюсь: предполагается, что вы уже заполнили базу данных с помощью последнего написанного нами метода. Далее в общих чертах изложено, как мы собираемся выполнять выборку в фоновом контексте.

1. Создаем фоновый контекст с помощью метода-инициализатора `initWithConcurrencyType:`, относящегося к классу `NSManagedObjectContext`, затем передаем этому методу значение `NSPrivateQueueConcurrencyType`. В результате получаем контекст, имеющий собственную закрытую очередь диспетчеризации. Поэтому, если вызвать в контексте блок `performBlock:`, этот блок будет выполнен в закрытой фоновой очереди.
2. Затем мы собираемся задать в фоновом контексте значение свойства `persistentStoreCoordinator`, которое будет равно экземпляру координатора нашего постоянного хранилища данных. Таким образом мы свяжем фоновый контекст с координатором постоянного хранилища. В результате, если выполнить выборку в фоновом контексте, эта операция позволит получить данные прямо с диска или из любого другого места, где их может хранить координатор.

3. Выполняем в фоновом контексте вызов performBlock:, а затем даем запрос на выборку. В рамках этого запроса требуется найти в стеке Core Data всех людей, чей возраст относится к диапазону от 100 до 200. Подчеркиваю: реалистичность эксперимента нас в данном случае не волнует. Я хочу лишь продемонстрировать, как действует выборка данных в фоновом режиме. Создавая запрос выборки данных, мы устанавливаем его свойство resultType в значение NSManagedObjectIDResultType. Так мы гарантируем, что результаты, возвращаемые после выполнения этого запроса на выборку, состоят не из управляемых объектов как таковых, а только из ID этих объектов. Как объяснялось ранее, мы не собираемся выбирать сами управляемые объекты, поскольку при выборке этих объектов в фоновом контексте не сможем использовать их в основном потоке. Итак, в фоновом контексте мы только выбираем их ID, а преобразуем эти ID в реальные управляемые объекты уже в главном контексте. После этого такие объекты можно использовать в основном потоке.

   Вот как создается запрос на выборку:

```
- (NSFetchRequest *) newFetchRequest{

  NSFetchRequest *request = [[NSFetchRequest alloc]
                             initWithEntityName:
  NSStringFromClass([Person class])];

  request.fetchBatchSize = 20;
  request.predicate =
  [NSPredicate predicateWithFormat:@"(age >= 100) AND (age <= 200)"];

  request.resultType = NSManagedObjectIDResultType;
  return request;

}
```

А вот как мы будем создавать фоновый контекст и выполнять в нем запрос на выборку данных:

```
- (BOOL) application:(UIApplication *)application
didFinishLaunchingWithOptions:(NSDictionary *)launchOptions{

  __weak NSManagedObjectContext *mainContext = self.managedObjectContext;
  __weak AppDelegate *weakSelf = self;
  __block NSMutableArray *mutablePersons = nil;

  /* Создаем фоновый контекст */
  NSManagedObjectContext *backgroundContext =
  [[NSManagedObjectContext alloc]
   initWithConcurrencyType:NSPrivateQueueConcurrencyType];

  backgroundContext.persistentStoreCoordinator =
  self.persistentStoreCoordinator;

  /* Выполняем блок в фоновом контексте */
```

```
  [backgroundContext performBlock:^{

    NSError *error = nil;
    NSArray *personIds = [backgroundContext
                          executeFetchRequest:[weakSelf newFetchRequest]
                          error:&error];

    if (personIds != nil && error == nil){
      mutablePersons = [[NSMutableArray alloc]
                        initWithCapacity:personIds.count];

      /* Теперь переходим в главный контекст и получаем эти объекты
      в главном контексте, исходя из их ID */
      dispatch_async(dispatch_get_main_queue(), ^{
        for (NSManagedObjectID *personId in personIds){
          Person *person = (Person *)[mainContext
                                      objectWithID:personId];
          [mutablePersons addObject:person];
        }
        [weakSelf processPersons:mutablePersons];
      });
    } else {
      NSLog(@"Failed to execute the fetch request.");
    }
  }];

  self.window = [[UIWindow alloc]
                 initWithFrame:[[UIScreen mainScreen] bounds]];

  self.window.backgroundColor = [UIColor whiteColor];
  [self.window makeKeyAndVisible];
  return YES;
}
```

Этот код собирает все управляемые объекты в виде массива, а затем вызывает в делегате нашего приложения метод `processPersons:`, обрабатывающий результаты массива. Напишем этот метод следующим образом:

```
- (void) processPersons:(NSArray *)paramPersons{
  for (Person *person in paramPersons){
    NSLog(@"First name = %@, last name = %@, age = %ld",
          person.firstName,
          person.lastName,
          (long)person.age.integerValue);
  }
}
```

## См. также

Разделы 7.4, 16.4 и 16.6.

# 16.10. Использование специальных типов данных в модели Core Data

## Постановка задачи

Вы считаете, что набор типов данных, представленных в Core Data, не удовлетворяет стоящим перед вами требованиям. Вам хотелось бы использовать в объектах моделей и дополнительные типы данных, например `UIColor`. Но такие объекты не содержатся в Core Data в готовом виде.

## Решение

Используйте преобразуемые типы данных.

## Обсуждение

Core Data позволяет создавать для объектов моделей свойства, а потом присваивать этим свойствам типы данных. Выбор при этом довольно ограничен: Core Data допускает использование лишь таких типов данных, которые могут быть преобразованы в экземпляр `NSData` и обратно. Но существует целый ряд популярных классов, которые вы по умолчанию не можете использовать в таких свойствах. Что же делать? Применяйте *преобразуемые свойства*. Сначала поясню, что это такое.

Допустим, вы хотите создать в Core Data объект модели и назвать этот объект `Laptop`. У данного объекта будет два свойства: `model` типа `String` и `color`, которое должно относиться к типу `UIColor`. В Core Data не предоставляется такой тип данных, поэтому для его получения нам придется создать подкласс от `NSValueTransformer`. Назовем этот класс `ColorTransformer`. Вот что станем делать при его реализации.

1. Переопределим его метод класса `allowsReverseTransformation` и вернем от него значение `YES`. Так мы сообщим Core Data о возможности преобразования цветов в данные и обратно.
2. Переопределим метод `transformedValueClass` этого класса и возвратим от него имя класса `NSData`. Возвращаемое значение этого метода сообщает Core Data, в какой класс вы будете преобразовывать специальное значение. В данном случае происходит преобразование `UIColor` в `NSData`. Поэтому вы должны вернуть от этого метода имя класса `NSData`.
3. Переопределим метод экземпляра `transformedValue:`, относящийся к преобразователю. В нашем методе будем брать входящее значение (которое в данном случае будет экземпляром `UIColor`), преобразовывать его в `NSData` и возвращать эти данные.
4. Переопределим метод экземпляра `reverseTransformedValue:`, относящийся к преобразователю, и сделаем это с совершенно противоположной целью. Берем входящее значение (здесь — данные) и преобразуем его в цвет.

Имея всю эту информацию, продолжим реализацию преобразователя. Чтобы сохранять цвет как данные, просто разделим его на целочисленные компоненты, которые будут сохраняться в массиве:

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

@implementation ColorTransformer

+ (BOOL) allowsReverseTransformation{
  return YES;
}

+ (Class) transformedValueClass{
  return [NSData class];
}

- (id) transformedValue:(id)value{

  /* Преобразуем цвет в данные */

  UIColor *color = (UIColor *)value;

  CGFloat red, green, blue, alpha;
  [color getRed:&red green:&green blue:&blue alpha:&alpha];
  CGFloat components[4] = {red, green, blue, alpha};
  NSData *dataFromColors = [[NSData alloc] initWithBytes:components
                             length:sizeof(components)];
  return dataFromColors;
}

- (id) reverseTransformedValue:(id)value{
  /* Выполняем обратное преобразование из данных в цвет */

  NSData *data = (NSData *)value;
  CGFloat components[4] = {0.0f, 0.0f, 0.0f, 0.0f};
  [data getBytes:components length:sizeof(components)];

  UIColor *color = [UIColor colorWithRed:components[0]
                    green:components[1]
                    blue:components[2]
                    alpha:components[3]];
  return color;
}
@end
```

Теперь возвращаемся к модели данных. Создадим управляемый объект `Laptop` и его атрибуты/свойства. Убедитесь, что атрибут `color` является преобразуемым. Выделив этот атрибут, нажмите на клавиатуре Alt+Command+3 и откройте Model Inspector (Инспектор модели) для этого атрибута. В поле name преобразуемого

класса введите имя специального преобразователя. В данном случае это будет `ColorTransformer` (рис. 16.21).

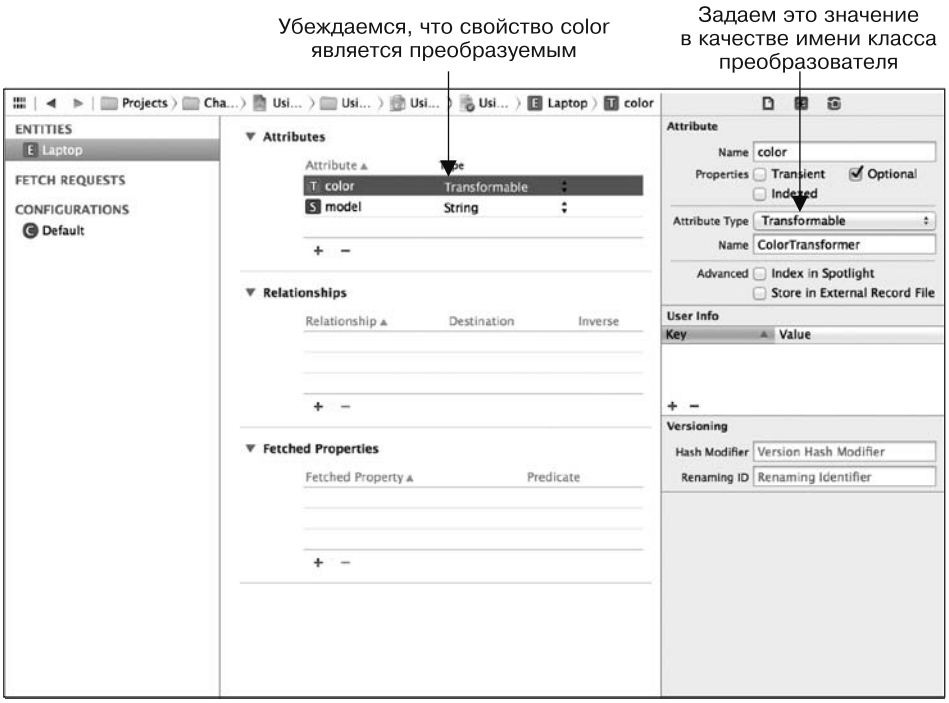


**Рис. 16.21.** Создание модели с преобразуемым атрибутом

Теперь воспользуемся приемами, изученными в разделе 16.2, и сгенерируем файл класса для управляемого объекта `Laptop`. После этого перейдем к заголовочному файлу этого управляемого объекта. Как видите, атрибут `color` рассматриваемого класса относится к типу `id`:

```
#import <Foundation/Foundation.h>
#import <CoreData/CoreData.h>

@interface Laptop : NSManagedObject

@property (nonatomic, retain) NSString * model;
@property (nonatomic, retain) id color;

@end
```

Уже неплохо. Но чтобы сделать код еще лучше, в частности помочь компилятору выявлять возможные проблемы (они могут возникать, если присваивать этому свойству значения неподходящих типов), вручную изменим этот тип данных на `UIColor`:

```
#import <Foundation/Foundation.h>
#import <CoreData/CoreData.h>
/* Обязательно импортируем эту информацию в таком виде, в каком
 UIColor находится в UIKit */
#import <UIKit/UIKit.h>

@interface Laptop : NSManagedObject

@property (nonatomic, retain) NSString * model;
@property (nonatomic, retain) UIColor *color;

@end
```

Итак, осталось объединить весь изученный материал и применить его на практике. В делегате нашего приложения создадим экземпляр `Laptop` и зададим для него красный цвет. Затем вставим этот объект в стек Core Data и попытаемся считать его обратно. Так мы проверим, удалось ли успешно сохранить цветовое значение и вновь достать его из базы данных:

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

@implementation AppDelegate

@synthesize managedObjectContext = _managedObjectContext;
@synthesize managedObjectModel = _managedObjectModel;
@synthesize persistentStoreCoordinator = _persistentStoreCoordinator;

- (BOOL) application:(UIApplication *)application
didFinishLaunchingWithOptions:(NSDictionary *)launchOptions{

  self.window = [[UIWindow alloc]
                 initWithFrame:[[UIScreen mainScreen] bounds]];
  self.window.backgroundColor = [UIColor whiteColor];
  [self.window makeKeyAndVisible];

  /* Сначала сохраняем объект laptop с заданным цветом */
  Laptop *laptop =
  [NSEntityDescription
   insertNewObjectForEntityForName:NSStringFromClass([Laptop class])
   inManagedObjectContext:self.managedObjectContext];

  laptop.model = @"model name";
  laptop.color = [UIColor redColor];

  NSError *error = nil;
  if ([self.managedObjectContext save:&error] == NO){
    NSLog(@"Failed to save the laptop. Error = %@", error);
  }

  /* Теперь находим этот же laptop */
```

```
  NSFetchRequest *fetch =
  [[NSFetchRequest alloc]
   initWithEntityName:NSStringFromClass([Laptop class])];
  fetch.fetchLimit = 1;

  fetch.predicate = [NSPredicate predicateWithFormat:@"color == %@",
  [UIColor redColor]];

  error = nil;
  NSArray *laptops = [self.managedObjectContext
                      executeFetchRequest:fetch
                      error:&error];

  /* Проверка на 1, поскольку лимит выборки равен 1 */
  if (laptops.count == 1 && error == nil){

    Laptop *fetchedLaptop = laptops[0];

    if ([fetchedLaptop.color isEqual:[UIColor redColor]]){
      NSLog(@"Right colored laptop was fetched");
    } else {
      NSLog(@"Could not find the laptop with the given color.");
    }
  }
  else {
    NSLog(@"Could not fetch the laptop with the given color. \
    Error = %@", error);
  }

return YES;
}
```

## См. также

Раздел 16.1.

# 17 Графика и анимация

## 17.0. Введение

Не сомневаюсь, что вам доводилось видеть программы для iPhone и iPad с очень красивой графикой. Кроме того, вы, наверное, встречали забавную анимацию в играх и других программах. При совместном использовании среды времени исполнения iOS и фреймворков программирования Cocoa Touch можно создавать самые разнообразные графические и анимационные эффекты с помощью сравнительно простого кода. Разумеется, качество этой графики и анимации частично зависит от эстетического вкуса программиста и его коллег-художников. Но в этой главе вы увидите, как много можно сделать в области графики и анимации, обладая весьма скромными навыками программирования.

Я не буду углубляться здесь в концептуальные базовые вопросы и расскажу о таких понятиях, как цветовые пространства, преобразования и графический контекст по ходу дела. Мы быстро рассмотрим некоторые фундаментальные вещи и почти сразу перейдем к коду.

В Cocoa Touch приложение состоит из *окон* (Window) и *видов* (View). Если у приложения есть пользовательский интерфейс, то в нем присутствует как минимум одно окно. Окно, в свою очередь, может содержать один или несколько видов. В Cocoa Touch окно является экземпляром класса `UIWindow`. Обычно в приложении открывается главное окно, и программист добавляет в это окно виды, представляющие разные компоненты пользовательского интерфейса. Видами являются, в частности, кнопки, подписи, изображения и специальные элементы управления, создаваемые самим программистом (Custom Controls). Отрисовка всех этих элементов пользовательского интерфейса и управление ими обеспечиваются во фреймворке UIKit.

Возможно, некоторые из этих вещей сложно понять сразу, но не волнуйтесь — по мере чтения главы вы постепенно разберетесь во всем. Особенно после знакомства с примерами, которые ждут нас впереди.

Apple предоставляет разработчикам мощные фреймворки, предназначенные для управления графикой и анимацией в операционных системах iOS и OS X. Далее перечислены некоторые из этих фреймворков и технологий.

- UIKit — это высокоуровневый фреймворк, позволяющий разработчикам создавать виды, окна, кнопки и другие компоненты пользовательского интерфейса. Кроме того, он включает ряд низкоуровневых API в состав высокоуровневого API, работать с которым довольно несложно.
- Quartz 2D — это основной движок, который работает «под капотом» операционной системы и обеспечивает отрисовку в iOS. Quartz применяется и во фреймворке UIKit.
- Core Graphics — фреймворк, поддерживающий графический контекст (подробнее об этом — в дальнейшем), загружающий изображения, отрисовывающий изображения и т. д.
- Core Animation — как следует из его названия, этот фреймворк обеспечивает применение анимации в iOS.

Приступая к рисованию на экране, исключительно важно усвоить одну концепцию: понять соотношение между точками и пикселами. С пикселами все ясно, но вот что такое *точки*? Это не зависящий от устройства аналог пикселов. Проще говоря, когда вы пишете приложение для iOS и от вас требуется указать какие-либо параметры, например высоту и ширину, то iOS считывает предоставленные вами значения как точки, а не как пикселы.

Например, если вы хотите заполнить весь экран на iPhone 5, ваш экранный элемент должен иметь ширину 320 и высоту 568. Однако мы знаем, что истинное разрешение экрана у iPhone 5 составляет 640 × 1136. В этом и заключается прелесть точек: оказывается, при работе с ними учитывается *коэффициент масштабирования*.

Здесь необходимо подробнее объяснить, что такое коэффициент масштабирования. Это обычное число с плавающей точкой, позволяющее iOS определить точное количество пикселов на экране. Для этого проверяется число логических точек, которые можно отобразить на этом экране. На iPhone 5 коэффициент масштабирования равен 2,0. Соответственно, iOS умножает 320 на 2, чтобы получить точное количество пикселов, которое устройство может отобразить по горизонтали, и умножает 568 на 2, чтобы получить количество пикселов по вертикали.

На экране устройства с iOS начало координат расположено в левом верхнем углу. Такие экраны также именуются ULO-экранами (от английского термина Upper Left Origin — «начало в левом верхнем углу»).

Это означает, что точка с координатами (0; 0) — крайняя точка в левом верхнем углу экрана. В таком случае положительные значения по оси *X* идут от нее направо, а положительные значения по оси *Y* — вниз. Иными словами, точка с координатой $x = 20$ находится на экране правее, чем точка с координатой $x = 10$. По оси *Y* точка с координатой $y = 20$ расположена ниже, чем точка $y = 10$.

В этой главе мы будем использовать объекты-виды типа `UIView` для отрисовки фигур, строк и любых других элементов, видимых на экране.

Предполагается, что вы работаете с новейшей версией Xcode on Apple. Если нет, откройте OS X, скачайте и установите новейшую версию Xcode.

Чтобы включить некоторые рассматриваемые здесь примеры кода в приложение, я сначала покажу, что нужно сделать для создания нового проекта в Xcode и подкласса от `UIView`, куда мы сможем поместить наш код.

1. Откройте Xcode.
2. В меню File (Файл) выполните команду New ▸ Project (Новый ▸ Проект).
3. Убедитесь, что в левой части экрана выбрана категория iOS. В этой категории укажите вариант Application (Приложение) (рис. 17.1)
4. В правой части экрана выберите Single View Application (Приложение с единственным видом) и нажмите Next (Далее) (рис. 17.1).

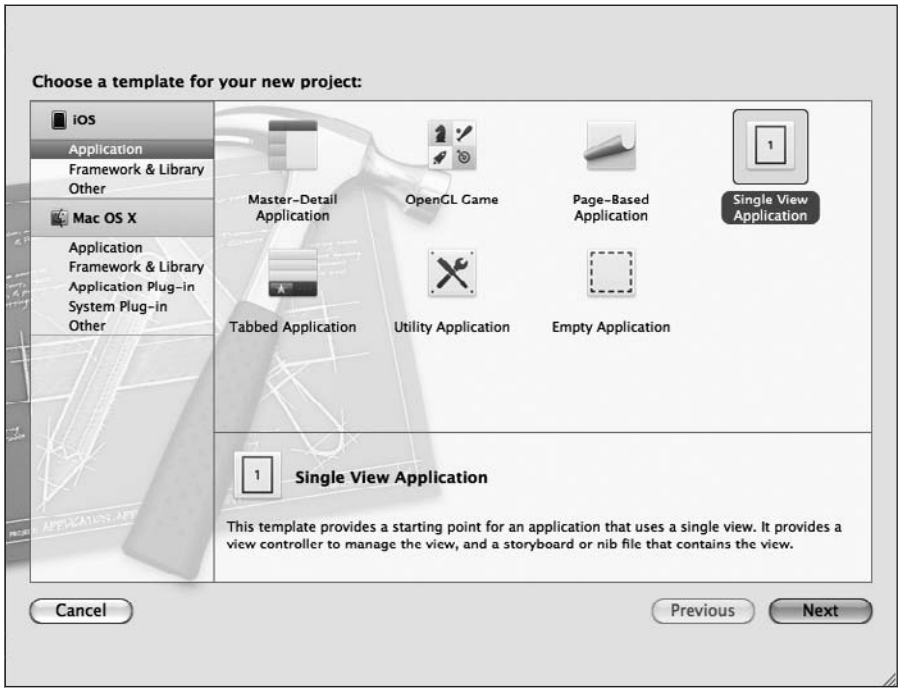


**Рис. 17.1.** Создание приложения с единственным видом для iOS в Xcode

5. В поле Product Name (Название продукта) (рис. 17.2) наберите имя вашего проекта.
6. В поле Company Identifier (Идентификатор компании) введите префикс, идентифицирующий пакет, который предшествует выбранному вами названию продукта. Обычно идентификатор записывается в формате `com.company`.
7. Из списка Devices (Устройства) выберите Universal, а затем нажмите кнопку Next (Далее).

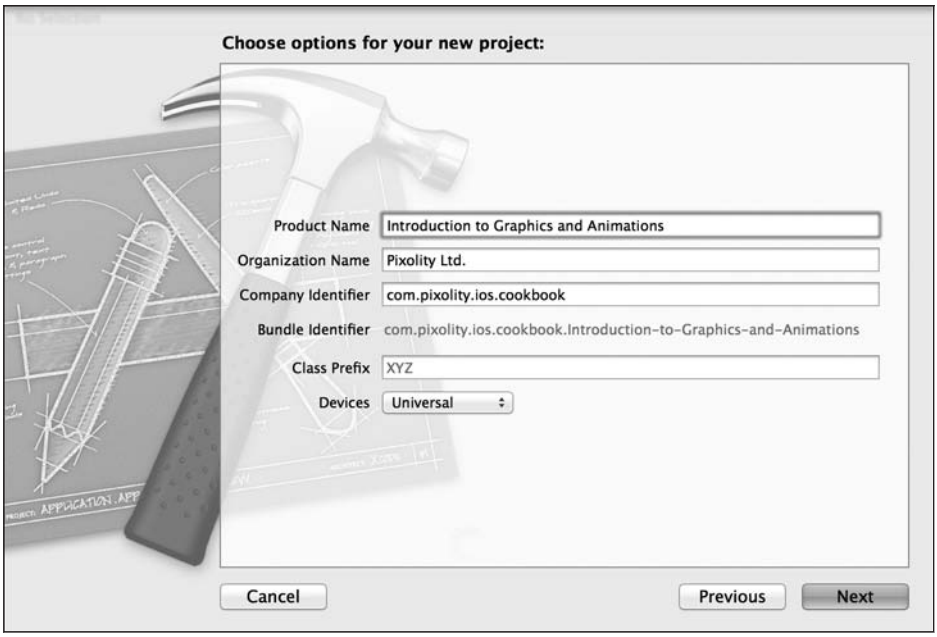


**Рис. 17.2.** Установка параметров для нового проекта в Xcode

8. В следующем окне выберите, где вы хотите сохранить ваш проект, и нажмите Create (Создать).

Теперь проект Xcode открыт. В левой части окна Xcode раскройте группу Graphics (Графика) и просмотрите все файлы, которые Xcode создала для проекта. Теперь потребуется создать объект вида для контроллера вида. Для этого выполните следующие шаги.

1. Щелкните правой кнопкой мыши на корневом каталоге группы вашего проекта в Xcode и выберите New File (Новый файл).
2. Убедитесь, что в диалоговом окне New File (Новый файл) слева в качестве категории указан вариант iOS, и в качестве подкатегории выберите Cocoa Touch (рис. 17.3)
3. В правой части окна выберите класс Objective-C, а потом нажмите Next (Далее) (см. рис. 17.3).
4. Убедитесь, что в следующем окне (рис. 17.4) в поле Subclass of (Подкласс) написано `UIView`, а потом задайте для вашего класса имя `View`. Далее сохраните файл на диске и нажмите Next (Далее).
5. Теперь откройте ваш файл раскадровки для iPhone и выберите вид для контроллера вида. Раскройте раздел Utilities (Вспомогательная область) в конструкторе интерфейсов и измените имя класса того вида, в котором находится ваш контроллер вида, на `View` (рис. 17.5).

Поскольку мы создали универсальное приложение, те же манипуляции понадобится выполнить в файле раскадровки для iPad. Обычно два этих файла называются `Main_iPhone.storyboard` и `Main_iPad.storyboard`.

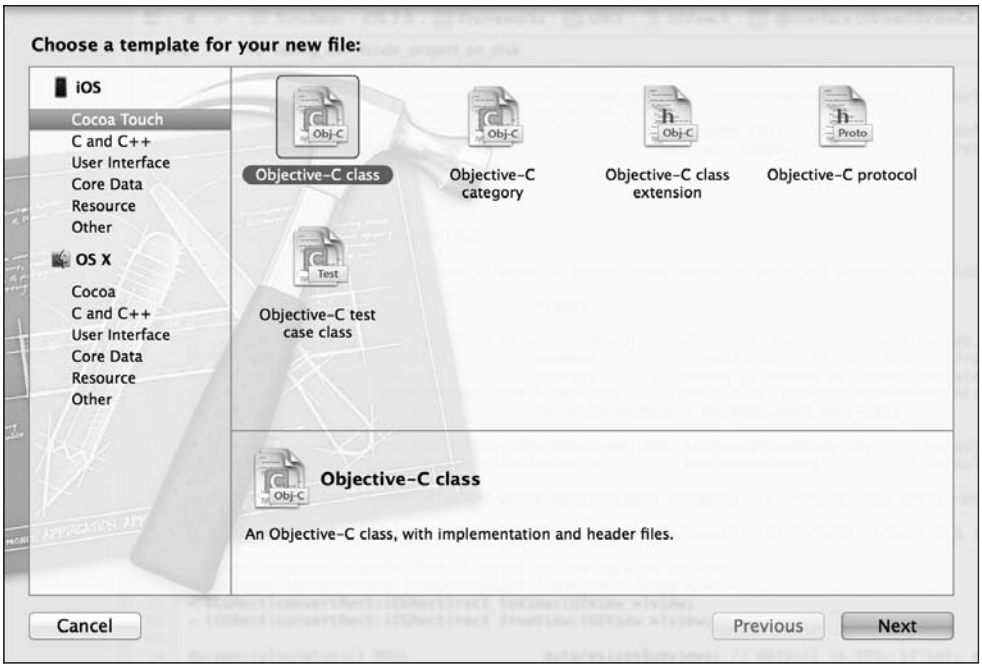


**Рис. 17.3.** Создание нового класса Objective-C в Xcode

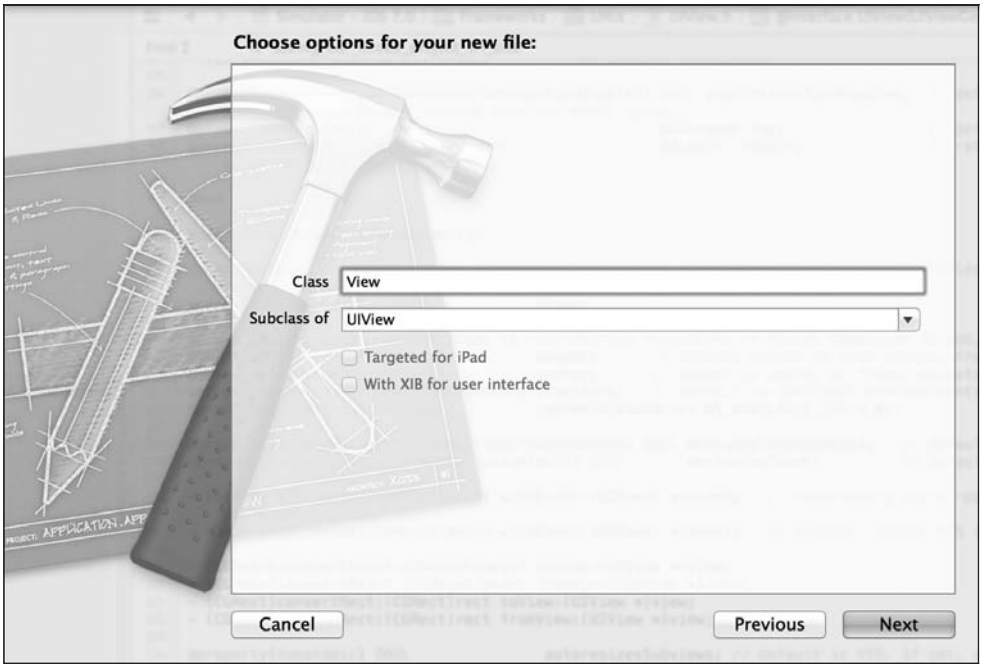


**Рис. 17.4.** Создание подкласса от UIView

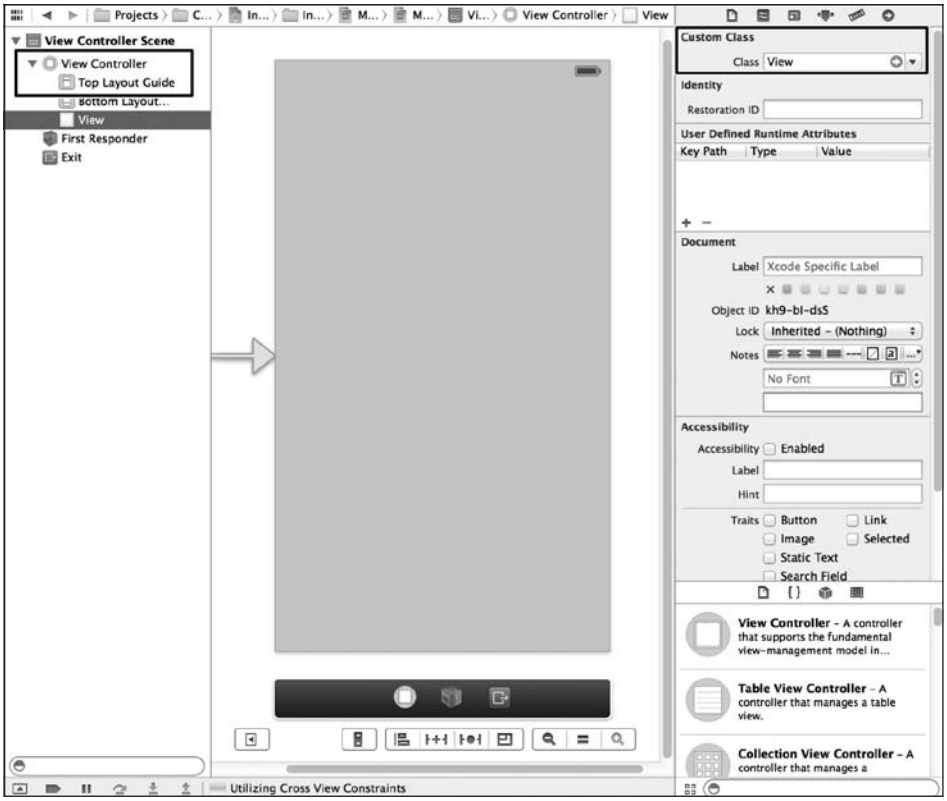


**Рис. 17.5.** Изменение имени класса контроллера вида в раскадровке

Итак, мы готовы приступить к написанию кода. А ведь сделать пришлось не так уж много — просто создать класс вида, относящегося к типу `UIView`, чтобы позже можно было изменять код этого класса. Потом мы воспользовались конструктором интерфейсов, чтобы задать в качестве класса вида контроллера вида тот самый объект вида, который мы создали ранее. Это означает, что вид контроллера вида теперь будет экземпляром созданного нами класса `View`.

Полагаю, вы уже просмотрели содержимое объекта-вида, сгенерированного Xcode. Один из самых важных методов этого объекта — `drawRect:`. Cocoa Touch автоматически вызывает этот метод всякий раз, когда приходит время отрисовывать вид. Данный метод используется для того, чтобы приказать объекту-виду отрисовать свое содержимое в графическом контексте. В свою очередь, Cocoa Touch автоматически готовит такой контекст для вида. Графический контекст можно сравнить с холстом (Canvas). Он предлагает огромное количество свойств, в частности цвет кисти (Pen Color), ее толщину (Pen Thickness) и т. д. Имея контекст, вы можете начать *рисовать* прямо внутри метода `drawRect:`, а Cocoa Touch гарантирует, что атрибуты и свойства контекста будут применены к вашим рисункам. Мы подробнее обсудим эти детали в дальнейшем, а пока перейдем к более интересным темам.

# 17.1. Перечисление и загрузка шрифтов

## Постановка задачи

Требуется использовать шрифты, предустановленные на устройстве с iOS, чтобы отобразить на экране какой-либо текст.

## Решение

Воспользуйтесь классом `UIFont`.

## Обсуждение

Шрифты имеют фундаментальное значение для отображения текста в графическом пользовательском интерфейсе. Во фреймворке UIKit программисту предоставляются высокоуровневые API, обеспечивающие перечисление, загрузку и использование шрифтов. В Cocoa Touch шрифты заключены в классе `UIFont`. В каждом устройстве с iOS есть встроенные системные шрифты. Шрифты распределены по *семействам* (Family), а в каждом семействе есть *гарнитуры* (Faces). Например, Helvetica — это семейство шрифтов, а Helvetica Bold — одна из гарнитур этого семейства. Чтобы шрифт можно было загрузить, необходимо знать гарнитуру шрифта (фактически его название), а чтобы узнать гарнитуру, нужно знать семейство. Итак, для начала перечислим все семейства шрифтов, которые уже установлены на устройстве, воспользовавшись методом `familyNames` класса `UIFont`:

```
- (void) enumerateFonts{

  for (NSString *familyName in [UIFont familyNames]){
    NSLog(@"Font Family = %@", familyName);
  }

}
```

Запустив эту программу в симуляторе, я получил примерно такие результаты:

```
Font Family = Thonburi
Font Family = Academy Engraved LE
Font Family = Snell Roundhand
Font Family = Avenir
Font Family = Marker Felt
Font Family = Geeza Pro
Font Family = Arial Rounded MT Bo Font Family = Trebuchet MS
...
```

Выстроив такой список семейств шрифтов, мы можем перечислить гарнитуры в каждом семействе. Будем пользоваться методом `fontNamesForFamilyName:` класса `UIFont`, а в ответ получим массив названий гарнитур из того семейства шрифтов, которое мы указали как параметр:

```
- (void) enumerateFonts{

  for (NSString *familyName in [UIFont familyNames]){
    NSLog(@"Font Family = %@", familyName);
    for (NSString *fontName in
         [UIFont fontNamesForFamilyName:familyName]){
      NSLog(@"\t%@", fontName);

    }
  }

}
```

Запустив этот код в симуляторе iOS, получим:

```
Font Family = Thonburi Thonburi-Bold Thonburi
Font Family = Academy Eng AcademyEngravedLetPla
Font Family = Snell Round SnellRoundhand-Bold SnellRoundhand-Black SnellRoundhand
...
```

Итак, мы видим, что *Thonburi* — это семейство шрифтов, а *Thonburi-Bold* — одна из гарнитур этого семейства. Теперь, зная имя шрифта, мы можем загружать шрифты в объекты типа `UIFont` с помощью метода класса `fontWithName:size:`, относящегося к классу `UIFont`:

```
__unused UIFont *font = [UIFont fontWithName:@"Thonburi-Bold"
size:12.0f];
```

Если в результате работы метода класса fontWithName:size:, относящегося к классу UIFont, имеем nil, это означает, что найти шрифт с указанным именем не удалось. Убедитесь, что шрифт с заданным вами именем присутствует в системе. Для этого сначала перечислите все семейства шрифтов, а потом все названия гарнитур из каждого семейства.

Кроме того, можно воспользоваться методом экземпляра `systemFontOfSize:`, относящимся к классу `UIFont` (или его «жирным» аналогом, `boldSystemFontOfSize:`), для загрузки локальных системных шрифтов — где бы они ни находились — прямо на устройстве, где работает ваш код. Стандартный системный шрифт в iOS — Helvetica.

Загрузив шрифты, можете переходить к разделу 17.2. Там мы воспользуемся загруженными шрифтами для отрисовки текста в графическом контексте.

## См. также

Раздел 17.2.

# 17.2. Отрисовка текста

## Постановка задачи

Требуется рисовать текст на экране устройства с iOS.

## Решение

Воспользуйтесь методом `drawAtPoint:withFont:` класса `NSString`.

## Обсуждение

Для отрисовки текста можно воспользоваться очень удобными методами, входящими в состав класса `NSString`. Один из таких методов — `drawAtPoint:withFont:`. Но прежде чем продолжить работу, еще раз удостоверьтесь в том, что выполнили все инструкции из введения к этой главе. Теперь у вас должен быть объект-вид, являющийся подклассом от `UIView`. Он должен называться `GraphicsViewControllerView`. Откройте этот файл. Если закомментирован метод экземпляра `drawRect:`, относящийся к объекту-виду, то раскомментируйте его, чтобы включить этот метод в объект:

```
#import "View.h"

@implementation View

- (id)initWithFrame:(CGRect)frame{
  self = [super initWithFrame:frame];
  if (self) {
    // Код инициализации
  }
  return self;
}

- (void)drawRect:(CGRect)rect{

}

@end
```

Именно в методе `drawRect:` будет происходить все рисование, как мы указывали ранее. Здесь мы можем приступать к загрузке шрифта, а потом нарисовать на экране простую текстовую строку, которая будет начинаться на уровне 40 по оси *X* и на уровне 180 по оси *Y* (рис. 17.6):

```
- (void)drawRect:(CGRect)rect{

  UIFont *helveticaBold =  [UIFont fontWithName:@"HelveticaNeue-Bold
                                           size:40.0f];

  NSString *myString = @"Some String";

  [myString drawAtPoint:CGPointMake(40, 180)
               withFont:helveticaBold];

}
```

В этом коде мы просто загружаем жирный шрифт Helvetica (кегль 40) и рисуем с его помощью текст `Some String`, который начинается в точке (40; 180).


**Рис. 17.6.** Произвольная строка, нарисованная в графическом контексте вида

# 17.3. Создание, установка и использование цветов

## Постановка задачи

Требуется иметь возможность получать ссылки на цветовые объекты с последующим использованием этих объектов при рисовании различных форм в виде. К числу форм можно отнести текст, прямоугольники, треугольники и сегменты линий.

## Решение

Воспользуйтесь классом `UIColor`.

## Обсуждение

Во фреймворке UIKit программисту предоставляются высокоуровневые абстракции цветов, инкапсулированные в объекте `UIColor`. В этом классе имеются очень удобные методы класса, в частности `redColor`, `blueColor`, `brownColor` и `yellowColor`. Тем не менее, если вас интересует иной цвет, кроме тех, чьи параметры явно задаются как параметры этого метода класса `UIColor`, можно воспользоваться методом класса `colorWithRed:green:blue:alpha:`, относящимся к классу `UIColor`, и загрузить искомое цветовое значение. Возвращаемое значение этого метода относится к типу `UIColor`. Данный метод имеет следующие параметры:

- `red` — доля красного в конкретном оттенке. Это значение может находиться в диапазоне от `0.0f` до `1.0f`, где `0.0f` полностью исключает красный компонент, а `1.0f` дает максимально насыщенный темно-красный цвет;
- `green` — доля зеленого, смешиваемая с красным в цвете. Это значение также может находиться в диапазоне от `0.0f` до `1.0f`;

- `blue` — доля голубого, смешиваемая с красным и зеленым в цвете. Это значение также может находиться в диапазоне от `0.0f` до `1.0f`;
- `alpha` — матовость (непрозрачность) цвета. Это значение может находиться в диапазоне от `0.0f` до `1.0f`, где `1.0f` делает цвет полностью матовым, а `0.0f` — полностью прозрачным (иными словами, невидимым).

Имея объект типа `UIColor`, вы можете воспользоваться его методом экземпляра `set`, чтобы в текущем графическом контексте этот цвет использовался для рисования.

Можно применять метод класса colorWithRed:green:blue:alpha:, относящийся к классу UIColor, для загрузки основных цветов, например красного. Для этого параметру red просто сообщается значение 1.0f, а параметрам green и blue — значение 0.0f. Значение параметра alpha выбираете сами.

Взглянув на рис. 17.1, мы видим, что заданный по умолчанию цвет фона для созданного нами объекта-вида — серый, довольно некрасивый. Исправим это. Просто найдем метод экземпляра `viewDidLoad` контроллера вида `GraphicsViewController` и изменим фоновый цвет вида на белый, как показано здесь:

```
- (void)viewDidLoad{
  [super viewDidLoad];
  self.view.backgroundColor = [UIColor whiteColor];
}
```

Для отрисовки текста в текущем графическом контексте будем пользоваться методами экземпляра класса NSString, подробнее об этом — чуть позже.

Теперь загрузим в объект типа `UIColor` пурпурный цвет, а потом нарисуем в графическом контексте вида текст `I Learn Really Fast`, использовав для этого жирный шрифт Helvetica кегля 30 (о загрузке шрифтов рассказано в разделе 17.1):

```
- (void)drawRect:(CGRect)rect{

  /* Загружаем цвет. */
  UIColor *magentaColor =[UIColor colorWithRed:0.5f
                                         green:0.0f
                                          blue:0.5f
                                         alpha:1.0f];

  /* Задаем цвет в графическом контексте. */
  [magentaColor set];

  /* Загружаем шрифт. */
  UIFont *helveticaBold = [UIFont fontWithName:@"HelveticaNeue-Bold"
                                          size:30.0f];

  /* Строка, которую требуется отрисовать. */
  NSString *myString = @"I Learn Really Fast";

  /* Рисуем строку выбранным шрифтом.
```

```
    Цвет мы уже установили. */
  [myString drawAtPoint:CGPointMake(25, 190)

withAttributes:@{
NSFontAttributeName : helveticaBold
}];}
```

Результат показан на рис. 17.7.

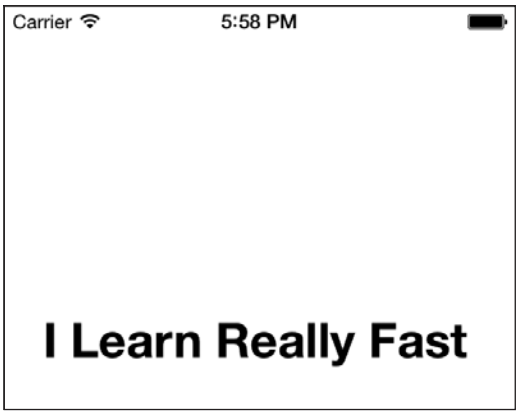


**Рис. 17.7.** Строка, отрисованная выбранным цветом в графическом контексте

Кроме того, мы можем воспользоваться методом экземпляра `drawInRect:withFont:`, относящимся к классу `NSString`, чтобы нарисовать текст внутри прямоугольной области. Текст будет растянут, чтобы он полностью занял отведенное пространство. Фреймворк UIKit даже позволяет переносить часть текста на следующую строку, если он не будет умещаться в отведенном прямоугольнике по горизонтали. Границы прямоугольной области инкапсулированы в структурах `CGRect`. Для создания границ прямоугольника можно использовать функцию `CGRectMake`:

```
- (void)drawRect:(CGRect)rect{

  /* Загружаем цвет. */
  UIColor *magentaColor = [UIColor colorWithRed:0.5f
                                          green:0.0f
                                           blue:0.5f
                                          alpha:1.0f];

  /* Задаем цвет в графическом контексте. */
  [magentaColor set];

  /* Загружаем шрифт. */
  UIFont *helveticaBold = [UIFont boldSystemFontOfSize:30];

  /* Строка, которую требуется отрисовать. */
  NSString *myString = @"I Learn Really Fast";
```

```
  /* Рисуем строку выбранным шрифтом.
     Цвет мы уже установили. */
  [myString drawInRect:CGRectMake(100,  /* x */
                                  120,  /* y */
                                  100,  /* ширина */
                                  200)  /* высота */

   options:NSStringDrawingUsesLineFragmentOrigin
   attributes:@{
     NSFontAttributeName : helveticaBold
   }
  context:nil];

}
```

Функция `CGRectMake` принимает четыре параметра:

- `x` — положение начала координат прямоугольника по оси *X* относительно графического контекста. В iOS это значение соответствует количеству точек, отсчитанному вправо от левой стороны прямоугольника;
- `y` — положение начала координат прямоугольника по оси *Y* относительно графического контекста. В iOS это значение соответствует количеству точек, отсчитанному вниз от верхней стороны прямоугольника;
- `width` — ширина прямоугольника в точках;
- `height` — высота прямоугольника в точках.

Результат выполнения кода показан на рис. 17.8.

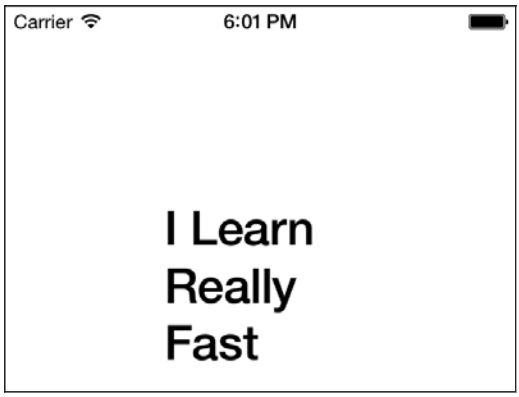


**Рис. 17.8.** Отрисовка строки в прямоугольном пространстве

`UIColor` — это, в сущности, предоставляемая в UIKit оболочка для класса `CGColor` из фреймворка Core Graphics. Переходя к довольно низкоуровневому программированию — то есть на уровне Core Graphics, — мы неожиданно обретаем значительно более полный контроль над использованием цветовых объектов и даже можем

определить цветовые компоненты, из которых состоит конкретный оттенок. Допустим, в каком-то другом коде вы получили объект типа `UIColor` и хотите определить, каковы содержащиеся в нем доли компонентов `red`, `green`, `blue` и `alpha`. Чтобы получить компоненты, из которых состоит объект `UIColor`, выполните следующие шаги.

1. Используйте метод экземпляра `CGColor`, относящийся к классу `UIColor`. В результате вы получите цветовой объект типа `CGColorRef`, представляющий собой ссылку на цветовой объект (Color Reference) — объект из фреймворка Core Graphics.
2. Применяйте функцию `CGColorGetComponents` для получения компонентов, составляющих цветовой объект.
3. При необходимости пользуйтесь функцией `CGColorGetNumberOfComponents`, чтобы определить количество компонентов, примененных для создания данного оттенка (красный + зеленый и т. д.).

   Вот пример:

```
- (void) drawRect:(CGRect)rect{
  /* Загрузка цвета */
  UIColor *steelBlueColor = [UIColor colorWithRed:0.3f
                                            green:0.4f
                                             blue:0.6f
                                            alpha:1.0f];

  CGColorRef colorRef = [steelBlueColor CGColor];

  const CGFloat *components = CGColorGetComponents(colorRef);

  NSUInteger componentsCount = CGColorGetNumberOfComponents(colorRef);

  NSUInteger counter = 0;
  for (counter = 0;
       counter < componentsCount;
       counter++){
    NSLog(@"Component %lu = %.02f",
          (unsigned long)counter + 1,
          components[counter]);
  }
}
```

После запуска кода получим в окне консоли следующий вывод:

```
Component 1 = 0.30
Component 2 = 0.40
Component 3 = 0.60
Component 4 = 1.00
```

## См. также

Раздел 17.1.

# 17.4. Отрисовка изображений

## Постановка задачи

Требуется возможность отрисовывать изображения на экране устройства с iOS.

## Решение

Используйте класс `UIImage` для загрузки изображения и относящийся к изображению метод `drawInRect:` для отрисовки картинки в графическом контексте.

## Обсуждение

Фреймворк UIKit очень упрощает задачи, связанные с рисованием. Все, что от вас требуется, — загрузить ваши изображения в экземпляры типа `UIImage`. В классе `UIImage` содержатся разнообразные методы класса и экземпляра, предназначенные для загрузки изображений. Вот некоторые из наиболее важных:

- `imageNamed:` (метод класса) — загружает изображение (если его удалось правильно загрузить, то и кэширует). Параметр этого метода — имя изображения в пакете, например `Tree Texture.png`;
- `imageWithData:` (метод класса) — загружает изображение из данных, инкапсулированных в экземпляре объекта `NSData`, который был передан данному методу в качестве параметра;
- `initWithContentsOfFile:` (метод экземпляра (для инициализации)) — использует указанный параметр как путь к изображению, которое должно быть загружено. Применяется для инициализации объекта изображения;

В данном случае подразумевается полный путь, указывающий на изображение в пакете приложения.

- `initWithData:` (метод экземпляра (для инициализации)) — использует полученный параметр типа `NSData` для инициализации изображения. Эти данные должны относиться к валидному изображению.

Чтобы добавить изображение в ваш проект в Xcode, выполните следующие шаги.

1. Найдите, где именно расположено изображение на вашем компьютере.
2. Перетащите это изображение в категорию изображений, которая обычно называется `images.xcassets`. Всю остальную работу Xcode выполнит за вас.

Для того чтобы получить ярлык Xcode, выполните следующие шаги:

1) найдите приложение Xcode в обозревателе;
2) находясь в обозревателе (Finder), нажмите на Xcode сочетание Command+I, чтобы получить информацию о приложении;

3) щелкните на ярлыке в верхнем левом углу окна справки Xcode;
4) нажмите Command+C, чтобы скопировать ярлык;
5) откройте приложение для предварительного просмотра (Preview);
6) нажмите сочетание клавиш Command+V, чтобы вставить ярлык Xcode в новое изображение;
7) полученный файл ICNS с пятью отдельными страницами сохраните в формате PDF, а потом удалите все, кроме ярлыка с наиболее высоким разрешением (страница 1 файла ICNS).

В этом разделе книги мы нарисуем изображение в графическом контексте, чтобы продемонстрировать общий принцип отрисовки изображений. Я уже нашел нужный файл и перетащил это изображение в мою программу для iOS. Теперь в пакете приложения есть изображение под названием Xcode.png (рис. 17.9).

**Рис. 17.9.** Ярлык Xcode, находящийся в приложении Xcode

Вот код для отрисовки изображения:

```
- (void)drawRect:(CGRect)rect{

  UIImage *image = [UIImage imageNamed:@"Xcode.png"];

  if (image != nil){
    NSLog(@"Successfully loaded the image.");
  } else {
    NSLog(@"Failed to load the image.");
  }

}
```

Если в пакете вашего приложения есть изображение `Xcode.png`, то после запуска этого кода на консоли появится надпись Successfully loaded the image (Изображение успешно загружено). Если изображения нет — будет написано Failed

to load the image (Не удалось загрузить изображение). В оставшейся части данного раздела предполагается, что у вас в пакете приложения есть нужное изображение. Можете смело помещать в пакет приложения и другие картинки, а потом ставить ссылки именно на них, а не на `Xcode.png`, которым я буду пользоваться в примерах кода.

Два самых простых способа отрисовки изображения типа `UIImage` в графическом контексте таковы:

- воспользоваться методом экземпляра `drawAtPoint:`, относящимся к классу `UIImage`. Таким образом в указанной точке отрисовывается изображение оригинального размера. Для создания этой точки используется функция `CGPointMake`;
- воспользоваться методом экземпляра `drawInRect:`, относящимся к классу `UIImage`. Изображение отрисовывается в заданной прямоугольной области. Для создания этой прямоугольной области используется функция `CGRectMake`:

```
- (void)drawRect:(CGRect)rect{

  /* Предполагается, что нужное изображение есть в пакете вашего приложения
     и его можно загрузить. */
  UIImage *xcodeIcon = [UIImage imageNamed:@"Xcode.png"];

  [xcodeIcon drawAtPoint:CGPointMake(0.0f,
                                     20.0f)];

  [xcodeIcon drawInRect:CGRectMake(50.0f,
                                   10.0f,
                                   40.0f,
                                   35.0f)];
}
```

При показанном ранее вызове `drawAtPoint:` будет отрисовано изображение оригинальных размеров с центром в точке (0; 20). При вызове `drawInRect:` будет отрисовано изображение с центром в точке (50; 10) размером 40 × 35 точек. Результаты показаны на рис. 17.10.

Соотношение сторон (Aspect Ratio) — это отношение между шириной и высотой изображения на экране компьютера. Предположим, у нас есть изображение размером 100 × 100 пикселов. Если нарисовать это изображение с началом координат в точке (0; 0) и задать для него размеры (100; 200), то вы сразу же заметите на экране, что картинка вытянулась по высоте (было 100 пикселов, стало 200). Метод экземпляра drawInRect:, относящийся к классу UIImage, оставляет на ваш выбор решение о том, как именно вы будете отрисовывать изображения. Иными словами, именно вы будете указывать значения *x*, *y*, ширины и высоты вашего изображения, определяя, как именно оно будет выглядеть на экране.

## См. также

Раздел 13.6.

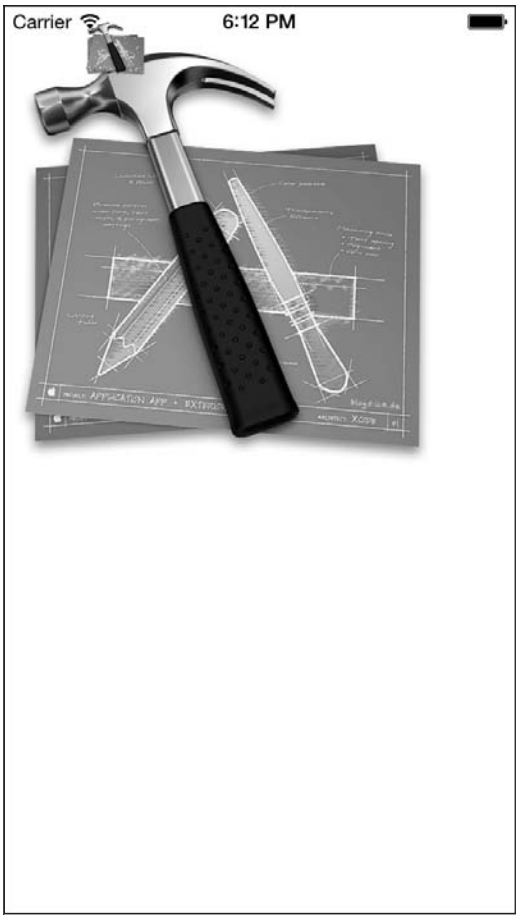


**Рис. 17.10.** Отрисовку изображения в графическом контексте можно выполнить с помощью двух различных методов

# 17.5. Создание адаптивных изображений

## Постановка задачи

Требуется экономить память и дисковое пространство, создавая для компонентов пользовательского интерфейса адаптивные изображения. Возможно, потребуется создать несколько вариантов одного и того же графического элемента, имеющих разные размеры. Например, это может быть несколько подобных кнопок, на каждой из которых используется одно и то же фоновое изображение.

Адаптивные изображения — это просто картинки в формате JPEG или PNG, которые можно загружать в экземпляры UIImage.

## Решение

Создайте адаптивное изображение, воспользовавшись методом экземпляра `resizableImageWithCapInsets:`, относящимся к классу `UIImage`.

## Обсуждение

На первый взгляд термин «адаптивное изображение» может показаться странным, но все становится на свои места, если учесть, что ваше приложение будет отображаться в довольно разных условиях, в зависимости от ситуации. Например, у вас может быть приложение для iOS, в котором все кнопки имеют фоновые изображения. Чем крупнее текст на кнопке, тем шире должна быть сама кнопка. Итак, есть два способа, которыми можно создать подходящие фоновые изображения для кнопок.

- Создать по одному изображению для каждого из размеров кнопки. В результате пакет приложения увеличится, возрастет потребление памяти, а вам придется выполнять больше работы. Кроме того, при изменении текста вновь потребуется подгонять изображение под размеры кнопки.
- Создать всего одно адаптивное изображение и использовать его во всем приложении для всех кнопок.

Несомненно, второй вариант кажется гораздо более привлекательным. Итак, что же представляют собой адаптивные изображения? Это просто изображения, состоящие из двух виртуальных областей:

- области, размер которой не меняется;
- области, размер которой свободно меняется и принимает нужные значения.

Как показано на рис. 17.11, мы создали изображение для кнопки. Внимательно рассмотрев это изображение, вы замечаете, что оно состоит из градиента. Область, которую я отрисовал вокруг прямоугольника, не может быть вырезана из приложения. Возникает вопрос: а почему? Смотрим еще внимательнее! Если я вырежу эту область и задам для нее значения высоты и ширины всего по 1 пикселу (как сейчас), то в приложении я смогу объединить сколько угодно таких однопиксельных полосок и сделать точно такую же область, какая выделена на этом рисунке (рис. 17.12).


**Рис. 17.11.** Изображение, в котором есть избыточная область, наиболее целесообразно сделать адаптивным


**Рис. 17.12.** Все отдельные срезы центральной секции изображения совершенно одинаковы

Итак, как нам уменьшить изображение, но по-прежнему иметь возможность создать из него кнопку? Ответ прост. В данном случае, когда изображение является совершенно одинаковым по всей длине, мы просто вырежем по центру изображения очень узкий фрагмент. Его ширина составит 1 пиксел, а высота не изменится. На рис. 17.13 показано, как изображение будет выглядеть после этой операции.

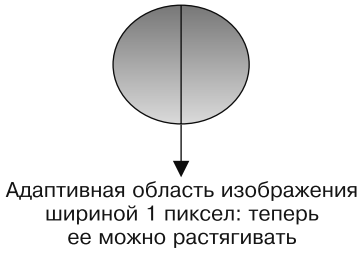


**Рис. 17.13.** Область изображения, размер которой можно изменять, теперь равна по ширине одной точке

И вот начинается самое интересное. Как мы можем сообщить iOS SDK, какие части изображения оставить нетронутыми, а какую часть растягивать? Оказывается, в iOS SDK уже предусмотрена такая возможность. Сначала загрузите ваше изображение в память с помощью API `UIImage`, изученных в этой главе. Создав экземпляр `UIImage` с таким изображением, которое гарантированно можно растягивать, преобразуйте этот экземпляр в адаптивное изображение. Это делается с помощью метода экземпляра `resizableImageWithCapInsets:`, относящегося как раз к рассматриваемому экземпляру. Параметр, принимаемый этим методом, относится к типу `UIEdgeInsets`, который, в свою очередь, определяется вот так:

```
typedef struct UIEdgeInsets {
  CGFloat top, left, bottom, right;
} UIEdgeInsets;
```

Краевые отступы нужны для того, чтобы создавать так называемые *девятичастные изображения* (nine-part images в терминологии Apple). Имеются в виду изображения, включающие в себя следующие девять компонентов:

- верхний левый угол;
- верхний край;
- верхний правый угол;
- правый край;
- нижний правый угол;
- нижний край;
- нижний левый угол;
- левый край;
- центр.

На рис. 17.14 все проиллюстрировано максимально наглядно.

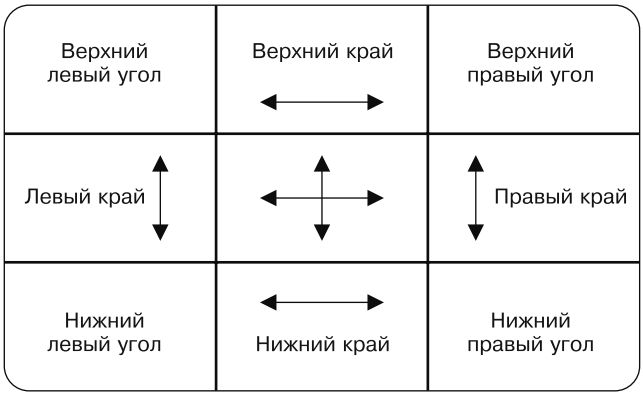


**Рис. 17.14.** Девятичастное изображение

Изображения сохраняются в девятичастном виде для того, чтобы программист мог адаптировать его размер по горизонтали и вертикали практически как угодно. Когда программисту требуется адаптировать изображение, некоторые из этих компонентов изменяют размер, а другие остаются нетронутыми. Никогда не меняются величины углов. Адаптация размеров других компонентов происходит следующим образом:

- *верхний край* — размер этого компонента изображения может изменяться по ширине, но не по высоте;
- *правый край* — размер этого компонента изображения может изменяться по высоте, но не по ширине;
- *нижний край* — размер этого компонента изображения, так же как и верхнего края, может изменяться по ширине, но не по высоте;
- *левый край* — размер этого компонента изображения, так же как и правого края, может изменяться по высоте, но не по ширине;
- *центр* — размеры центра могут меняться как по высоте, так и по ширине.

Значения отступов по верхнему, левому, нижнему и правому краям задают размеры той области, которую вы не хотите растягивать. Например, вы задали для левого края значение 10, для верхнего края — значение 11, для правого края — значение 14 и для нижнего — 5. Так вы приказываете iOS провести через изображение вертикальную *линию* в 10 точках от левого края, горизонтальную линию в 11 точках от верхнего края, еще одну вертикальную линию в 14 точках от правого края и, наконец, горизонтальную линию в 5 точках от нижнего края. Прямоугольная область, *заключенная* между этими линиями, может изменять размер (является адаптивной), а область вне этого контура — не может. Если все кажется немного запутанным, представьте себе прямоугольник (ваше изображение), а затем начертите внутри него другой прямоугольник. Размеры внутреннего прямоугольника могут меняться, размеры внешнего — нет. Предлагаю вновь рассмотреть ситуацию на картинке (рис. 17.15).

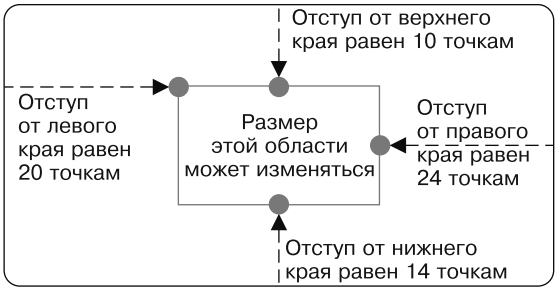


**Рис. 17.15.** Размеры изменяемой части изображения определяются по величине отступов от краев

На самом деле отступы слева и справа на рис. 17.15 одинаковы. Отступы от нижнего и верхнего краев также одинаковы. Я указал для них разные значения лишь для того, чтобы процесс создания отступов был более понятным и логичным. Если бы все значения отступов были одинаковы, то позже мы могли бы запутаться: а о каком из отступов сейчас идет речь?

Для такого изображения, как показано на рис. 17.15, краевые отступы создаются следующим образом:

```
UIEdgeInsets edgeInsets;
edgeInsets.left = 20.0f;
edgeInsets.top = 10.0f;
edgeInsets.right = 24.0f;
edgeInsets.bottom = 14.0f;
```

А теперь возвращаемся к учебному коду. Здесь мы попытаемся использовать адаптивное изображение, показанное на рис. 17.13, в реальном приложении. Мы создадим кнопку и поместим ее в центре единственного вида, находящегося в нашем контроллере вида. На кнопке будет написано Stretched Image on Button (Адаптивное изображение на кнопке). Кнопка будет иметь 200 точек в ширину и 44 точки в высоту. Вот наш код:

```
#import "ViewController.h"

@interface ViewController ()
```

```
@property (nonatomic, strong) UIButton *button;
@end

@implementation ViewController

- (void)viewDidLoad{
  [super viewDidLoad];

  /* Инстанцируем кнопку */
  self.button = [UIButton buttonWithType:UIButtonTypeCustom];
  [self.button setFrame:CGRectMake(0.0f, 0.0f, 200.0f, 44.0f)];

  /* Задаем надпись для кнопки */
  [self.button setTitle:@"Stretched Image on Button"
               forState:UIControlStateNormal];

  /* Корректируем шрифт для текста */
  self.button.titleLabel.font = [UIFont systemFontOfSize:15.0f];

  /* Создаем адаптивное изображение */
  UIImage *image = [UIImage imageNamed:@"Button"];
  UIEdgeInsets edgeInsets;
  edgeInsets.left = 14.0f;
  edgeInsets.top = 0.0f;
  edgeInsets.right = 14.0f;
  edgeInsets.bottom = 0.0f;
  image = [image resizableImageWithCapInsets:edgeInsets];

  /* Задаем фоновое изображение для кнопки */
  [self.button setBackgroundImage:image forState:UIControlStateNormal];

  [self.view addSubview:self.button];
  self.button.center = self.view.center;

}

@end
```

Теперь, запустив приложение, вы увидите примерно такую картинку, как на рис. 17.16.

## См. также

Раздел 17.4.

# 17.6. Отрисовка линий

## Постановка задачи

Требуется просто рисовать линии в графическом контексте.

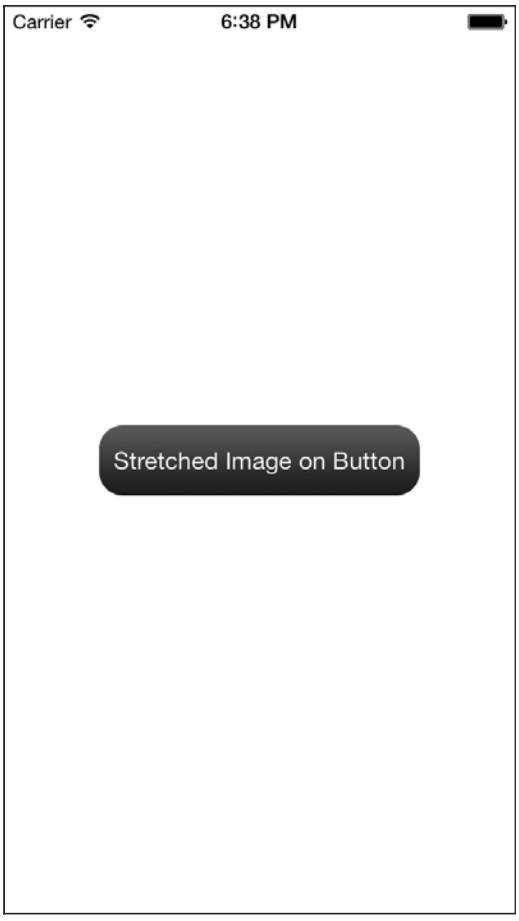


**Рис. 17.16.** На экране находится кнопка с адаптивным фоновым изображением

## Решение

Получите описатель для вашего графического контекста, а потом пользуйтесь функциями `CGContextMoveToPoint` и `CGContextAddLineToPoint` для отрисовки линии.

## Обсуждение

Когда мы говорим о рисовании фигур в iOS или OS X, мы подразумеваем *пути* (paths). Что такое путь в данном случае? Путь возникает между одной или несколькими сериями точек, расположенными на экране. Между путями и линиями существует значительная разница. Путь может содержать несколько линий, но линия не может содержать несколько путей. Считайте, что путь — это просто серия точек.

Линии нужно рисовать, пользуясь путями. Укажите начальную и конечную точки пути, а потом прикажите Core Graphics заполнить этот путь за вас. Core

Graphics считает, что вы создали линию вдоль этого пути, и нарисует его указанным вами цветом (см. раздел 17.3).

Более подробно мы рассмотрим пути в дальнейшем (в разделе 17.7), а пока сосредоточимся на том, как создавать с помощью путей прямые линии. Для этого нужно выполнить следующие шаги.

1. Выбрать цвет в вашем графическом контексте (см. раздел 17.3).
2. Получить описатель графического контекста — это делается с помощью функции `UIGraphicsGetCurrentContext`.
3. Задать начальную точку для линии, воспользовавшись процедурой `CGContextMoveToPoint`.
4. Переместить перо в графическом контексте, воспользовавшись процедурой `CGContextAddLineToPoint`, и указать конечную точку линии.
5. Создать намеченный путь с помощью процедуры `CGContextStrokePath`. Эта процедура отрисует путь в графическом контексте, использовав указанный вами цвет.

Кроме того, можно воспользоваться процедурой `CGContextSetLineWidth`, которая задает толщину линий, отрисовываемых в заданном графическом контексте. Первый параметр этой процедуры — графический контекст, на котором вы рисуете, а второй параметр — толщина линии, выражаемая числом с плавающей точкой (`CGFloat`).

В iOS толщина линии измеряется в логических точках.

Вот пример:

```
- (void)drawRect:(CGRect)rect{

  /* Задаем цвет, которым собираемся отрисовывать линию. */
  [[UIColor brownColor] set];

  /* Получаем актуальный графический контекст. */
  CGContextRef currentContext = UIGraphicsGetCurrentContext();

  /* Задаем толщину линии. */
  CGContextSetLineWidth(currentContext,
                        5.0f);

  /* В этой точке будет начинаться линия. */
  CGContextMoveToPoint(currentContext,
                       50.0f,
                       10.0f);

  /* В этой точке линия будет заканчиваться. */
  CGContextAddLineToPoint(currentContext,
                          100.0f,
                          200.0f);

  /* Для отрисовки линии используем цвет, заданный в контексте в настоящий
```

```
    момент. */
  CGContextStrokePath(currentContext);

}
```

Запустив этот код в симуляторе iOS, вы получите примерно такие результаты, как на рис. 17.17.

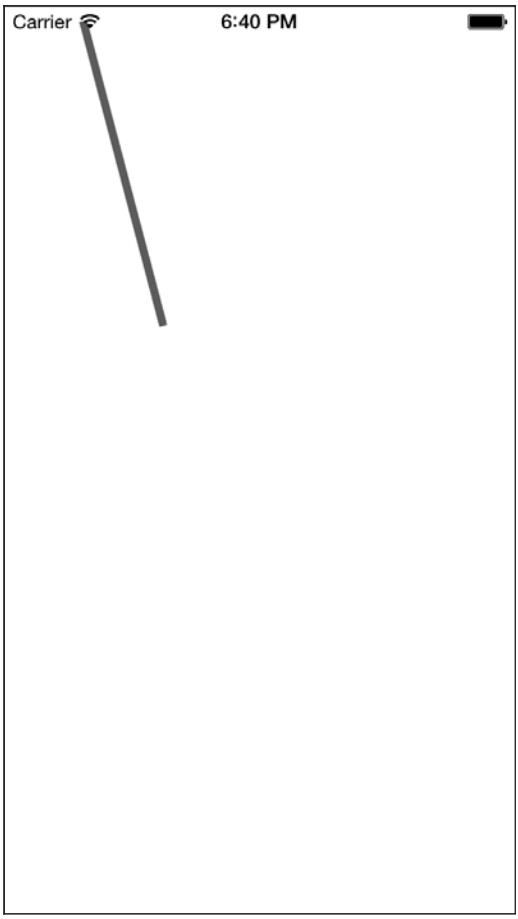


**Рис. 17.17.** Рисование линии в текущем графическом контексте

Приведу еще один пример. Как было упомянуто ранее, процедура `CGContextAddLineToPoint` указывает конечную точку данной линии. А что делать, если мы уже провели линию из точки (20; 20) в точку (100; 100), а теперь хотим провести линию из точки (100; 100) в точку (300; 100)? Может возникнуть версия, что, нарисовав первую линию, мы должны переместить перо в точку (100; 100) с помощью процедуры `CGContextMoveToPoint`, а потом провести линию в точку (300; 100), используя процедуру `CGContextAddLineToPoint`. Да, это сработает, но задачу можно решить более

эффективным способом. После того как вы вызовете процедуру CGContextAddLineToPoint для указания конечной точки отрисовываемой в данный момент линии, положение вашего пера изменится на значение, которое будет передано этому методу. Иными словами, после того, как вы выпустите метод, воспользовавшись пером, метод поставит перо в конечной точке того объекта, который был отрисован (объект может быть любым). Итак, чтобы нарисовать еще одну линию из актуальной конечной точки в новую точку, нужно просто еще раз вызвать процедуру CGContextAddLineToPoint, сообщив ей новую конечную точку. Вот пример:

```
- (void)drawRect:(CGRect)rect{

  /* Задаем цвет, которым мы собираемся отрисовывать линию. */
  [[UIColor brownColor] set];

  /* Получаем актуальный графический контекст. */
  CGContextRef currentContext = UIGraphicsGetCurrentContext();

  /* Задаем толщину линий. */
  CGContextSetLineWidth(currentContext,
                        5.0f);

  /* В этой точке будет начинаться линия. */
  CGContextMoveToPoint(currentContext,
                       20.0f,
                       20.0f);

  /* В этой точке линия будет заканчиваться. */
  CGContextAddLineToPoint(currentContext,
                          100.0f,
                          100.0f);

  /* Продолжаем линию до новой точки. */
  CGContextAddLineToPoint(currentContext,
                          300.0f,
                          100.0f);

  /* Для отрисовки линии используем цвет, заданный в контексте в настоящий
     момент. */
  CGContextStrokePath(currentContext);

}
```

Результат показан на рис. 17.18. Как видите, удалось успешно отрисовать обе линии, не перемещая перо для отрисовки второй линии.

Точка соединения двух линий называется перемычкой (Join). Работая с Core Graphics, можно указывать тип перемычки, которую вы хотите сделать между линиями, сочлененными друг с другом. Для выбора такого типа используется процедура CGContextSetLineJoin. Она принимает два параметра: во-первых, графический контекст, в котором вы задаете перемычку такого типа, а во-вторых, сам тип перемычки, CGLineJoin. CGLineJoin — это перечень следующих значений:

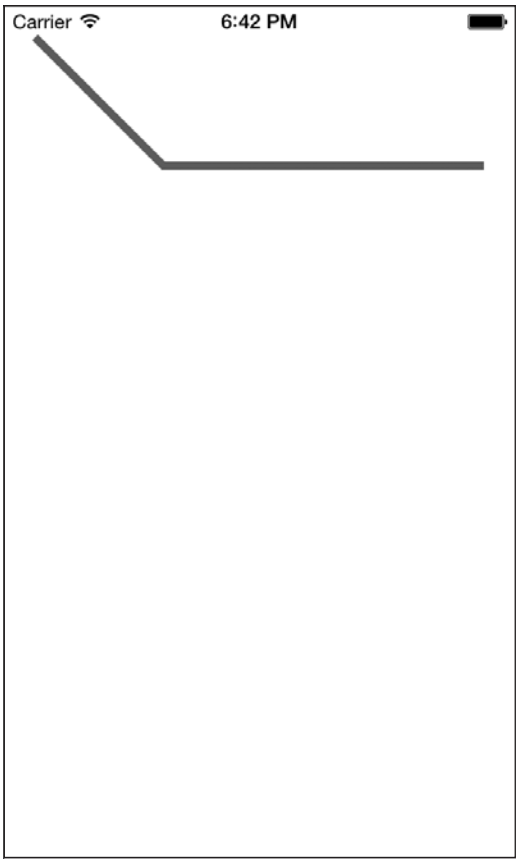


**Рис. 17.18.** Одновременно отрисовываем две линии

- `kCGLineJoinMiter` — на месте перемычки образуется острый угол. Этот тип задается по умолчанию;
- `kCGLineJoinBevel` — угол на месте перемычки линий будет немного спрямлен, как будто обтесан;
- `kCGLineJoinRound` — как понятно из названия, такая перемычка — скругленная.

Рассмотрим пример. Допустим, мы хотим написать программу, способную отрисовывать в графическом контексте «скатные крыши», каждая из которых иллюстрировала бы определенный тип перемычки между линиями, а также выводить под «крышей» текст с названием используемой перемычки. В результате получится рисунок, напоминающий рис. 17.19.

Для решения этой задачи я написал метод `drawRooftopAtTopPointof:textToDisplay:lineJoin:`, принимающий три параметра:

- точку, в которой должна располагаться верхушка «крыши»;
- текст, отображаемый под «крышей»;
- используемый тип перемычки.

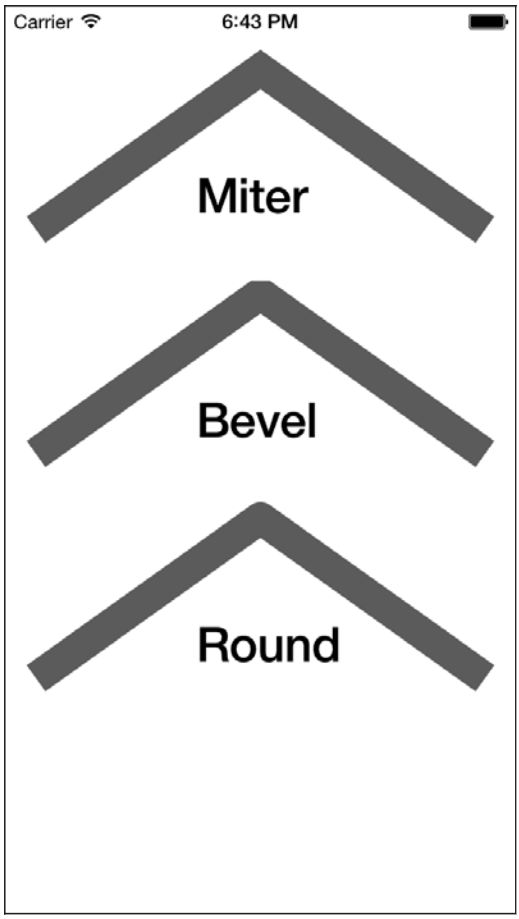


**Рис. 17.19.** Три типа перемычек между линиями, существующие в Core Graphics

Код будет таким:

```
- (void) drawRooftopAtTopPointof:(CGPoint)paramTopPoint
                  textToDisplay:(NSString *)paramText
                       lineJoin:(CGLineJoin)paramLineJoin{

  /* Задаем цвет, которым собираемся отрисовывать линию. */
  [[UIColor brownColor] set];

  /* Получаем актуальный графический контекст. */
  CGContextRef currentContext = UIGraphicsGetCurrentContext();

  /* Задаем перемычку между линиями. */
  CGContextSetLineJoin(currentContext,
                       paramLineJoin);

  /* Задаем толщину линий. */
```

```
  CGContextSetLineWidth(currentContext,
                        20.0f);

  /* В этой точке будет начинаться линия. */
  CGContextMoveToPoint(currentContext,
                       paramTopPoint.x - 140,
                       paramTopPoint.y + 100);

  /* В этой точке линия будет заканчиваться. */
  CGContextAddLineToPoint(currentContext,
                          paramTopPoint.x,
                          paramTopPoint.y);

  /* Продолжаем линию до новой точки, чтобы получилась фигура,
     напоминающая крышу. */
  CGContextAddLineToPoint(currentContext,
                          paramTopPoint.x + 140,
                          paramTopPoint.y + 100);

  /* Для отрисовки линии используем цвет, заданный в контексте в настоящий
     момент. */
  CGContextStrokePath(currentContext);

  /* Рисуем под крышей текст, при этом используется черный цвет. */
  [[UIColor blackColor] set];

  /* Теперь рисуем текст. */
  CGPoint drawingPoint = CGPointMake(paramTopPoint.x - 40.0f,
                                     paramTopPoint.y + 60.0f);
  [paramText drawAtPoint:drawingPoint
                withFont:[UIFont boldSystemFontOfSize:30.0f]];

}
```

А теперь вызовем наш метод в методе экземпляра `drawRect:` объекта-вида, где находится графический контекст:

```
- (void)drawRect:(CGRect)rect{

  [self drawRooftopAtTopPointof:CGPointMake(160.0f, 40.0f)
                  textToDisplay:@"Miter"
                       lineJoin:kCGLineJoinMiter];

  [self drawRooftopAtTopPointof:CGPointMake(160.0f, 180.0f)
                  textToDisplay:@"Bevel"
                       lineJoin:kCGLineJoinBevel];

  [self drawRooftopAtTopPointof:CGPointMake(160.0f, 320.0f)
                  textToDisplay:@"Round"
                       lineJoin:kCGLineJoinRound];
}
```

## См. также

Разделы 17.3 и 17.7.

# 17.7. Создание путей

## Постановка задачи

Необходимо иметь возможность нарисовать в графическом контексте любой желаемый контур.

## Решение

Создавайте и отрисовывайте пути.

## Обсуждение

Если расположить рядом серию точек, они могут образовать фигуру. Серия фигур, составленных вместе, образует путь. Управлять путями в Core Graphics очень удобно. В разделе 17.6 мы опосредованно работали с путями, пользуясь функциями `CGContext`. Но в Core Graphics есть и такие функции, которые работают с путями напрямую. Вскоре мы с ними познакомимся.

Пути относятся к тому графическому контексту, в котором они нарисованы. У путей нет границ либо конкретных контуров, в отличие от фигур, рисуемых по ним. Однако у путей есть ограничивающие рамки. Не забывайте, что граница и ограничивающая рамка — не тождественные понятия. Границы — это пределы, в которых вы можете рисовать на холсте, а ограничивающая рамка пути — это наименьший прямоугольник, в котором содержатся все фигуры, точки и другие объекты, отрисованные по данному конкретному пути. Пути можно сравнить с марками, а графический контекст — с конвертом для письма. Всякий раз, когда вы решите послать открытку другу, конверты для нее будут одинаковыми, но может различаться количество марок, которые вы наклеите на конверт (в нашем случае могут различаться пути).

Когда вы закончите рисование на определенном пути, можно отрисовать этот путь в графическом контексте. Разработчики, которым доводилось заниматься программированием игр, знакомы с понятием *буфера*. Буфер отрисовывает закрепленные за ним сцены и в нужный момент *сбрасывает* это содержимое на экран. Пути — это, в сущности, буферы. Они напоминают невидимые границы, рисуемые на холсте.

Приступая к непосредственной работе с путями, начнем с создания самого пути. Метод, создающий путь, возвращает описатель, которым вы будете пользоваться всякий раз, когда решите нарисовать что-либо на этом пути. Описатель передается Core Graphics для справки. Создав путь, вы сможете добавить к нему различные линии, фигуры и точки и только потом отрисовать его. Путь можно либо заполнить

определенным цветом заливки, либо отрисовать штрихами в графическом контексте. Вот методы, с которыми придется работать:

- `CGPathCreateMutable` (функция) — создает новый изменяемый путь типа `CGMutablePathRef` и возвращает его описатель. Как только мы закончим работу с этим путем, от него необходимо избавиться — об этом мы вскоре поговорим;
- `CGPathMoveToPoint` (процедура) — перемещает на путь актуальное положение пера. Перо оказывается в точке, заданной в параметре типа `CGPoint`;
- `CGPathAddLineToPoint` (процедура) — отрисовывает сегмент линии от актуальной позиции пера до указанной позиции (которая опять же указывается как значение типа `CGPoint`);
- `CGContextAddPath` (процедура) — добавляет заданный путь (на который указывает переданный здесь описатель) в графический контекст. Этот путь готов для рисования;
- `CGContextDrawPath` (процедура) — отрисовывает заданный путь в графическом контексте;
- `CGPathRelease` (процедура) — высвобождает память, выделенную для описателя пути;
- `CGPathAddRect` (процедура) — добавляет к пути прямоугольник. Границы прямоугольника указаны в структуре `CGRect`.

Существуют три важных рисовальных метода, выполнение которых можно задать процедуре `CGContextDrawPath`:

- `kCGPathStroke` — рисует линию (штрих), отмечающий границу или кромку пути. Штрих рисуется актуальным цветом, выбранным в данный момент;
- `kCGPathFill` — заполняет цветом заливки область, вокруг которой описан путь. Заливка выполняется в актуальном цвете, выбранном в данный момент;
- `kCGPathFillStroke` — комбинирует штрих и заливку. Для заполнения пути использует актуальный цвет заливки, а выбранный цвет штриха применяет для отрисовки пути. В следующем разделе будет рассмотрен пример использования этого метода.

Рассмотрим пример. Допустим, нам необходимо нарисовать голубую линию, идущую по экрану из верхнего левого угла в нижний правый, и другую линию, идущую из верхнего правого угла в левый нижний. Так мы нарисуем на экране большой крест, напоминающий букву «X».

В этом примере я удалил из приложения в симуляторе iOS статусную панель. Если вы не хотите с этим возиться, то можете сразу переходить к коду, приведенному далее. При наличии статусной панели результат выполнения кода будет лишь незначительно отличаться от того, что показано на моем скриншоте. Чтобы скрыть статусную панель, найдите в вашем проекте Xcode файл Info.plist и добавьте в этот файл ключ UIStatusBarHidden со значением YES (рис. 17.20). В таком случае сразу после открытия приложения статусная панель будет скрыта.

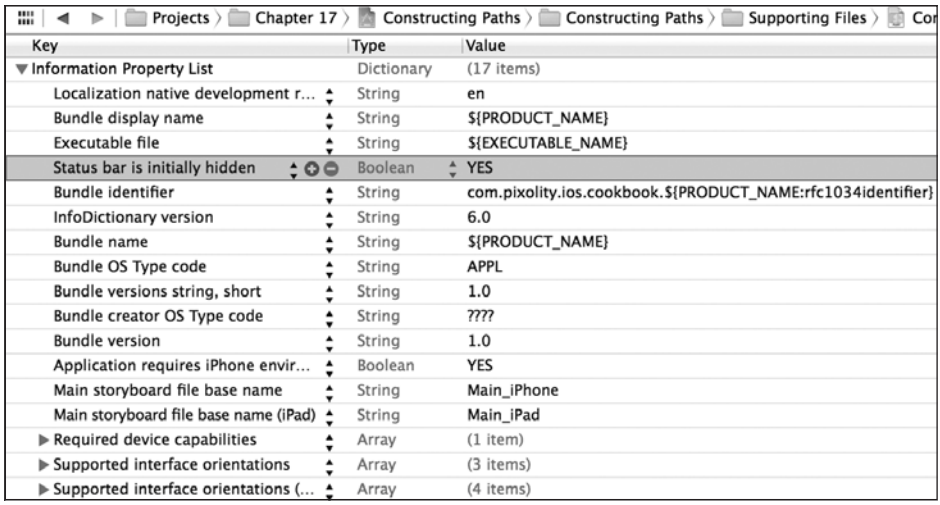


**Рис. 17.20.** Операция с файлом Info.plist, позволяющая скрыть статусную панель приложения iOS

```
- (void)drawRect:(CGRect)rect{

  /* Создаем путь. */
  CGMutablePathRef path = CGPathCreateMutable();

  /* Каковы размеры экрана? Мы хотим, чтобы X растянулся на весь экран. */
  CGRect screenBounds = [[UIScreen mainScreen] bounds];

  /* Начинаем с верхнего левого угла. */
  CGPathMoveToPoint(path,
                    NULL,
                    screenBounds.origin.x,
                    screenBounds.origin.y);

  /* Проводим линию из верхнего левого в нижний правый угол экрана. */
  CGPathAddLineToPoint(path,
                       NULL,
                       screenBounds.size.width,
                       screenBounds.size.height);

  /* Начинаем другую линию из верхнего правого угла. */
  CGPathMoveToPoint(path,
                    NULL,
                    screenBounds.size.width,
                    screenBounds.origin.y);

  /* Проводим линию из верхнего правого в нижний левый угол. */
  CGPathAddLineToPoint(path,
                       NULL,
```

```
                        screenBounds.origin.x,
                        screenBounds.size.height);

    /* Получаем контекст, в котором должен быть отрисован путь. */
    CGContextRef currentContext = UIGraphicsGetCurrentContext();

    /* Добавляем путь к контексту, чтобы позже его можно было отрисовать. */
    CGContextAddPath(currentContext,
                     path);

    /* Задаем для штриха голубой цвет. */
    [[UIColor blueColor] setStroke];

    /* Отрисовываем путь этим цветом. */
    CGContextDrawPath(currentContext,
                      kCGPathStroke);

    /* Наконец, высвобождаем объект пути. */
    CGPathRelease(path);

}
```

Параметр NULL, передаваемый таким процедурам, как CGPathMoveToPoint, представляет возможные преобразования, которые могут быть применены при отрисовке фигур и линий по заданному пути. Подробнее о преобразованиях рассказано в разделах 17.11–17.13.

Итак, нарисовать путь в графическом контексте очень просто. На самом деле следует всего лишь запомнить, как создать новый изменяемый путь (`CGPathCreateMutable`), добавить этот путь к вашему графическому контексту (`CGContextAddPath`) и отрисовать путь в графическом контексте (`CGContextDrawPath`). Запустив этот код, вы получите примерно такой результат, как на рис. 17.21.

## См. также

Разделы 17.6, 17.11–17.13.

# 17.8. Отрисовка прямоугольников

## Постановка задачи

Требуется отрисовывать прямоугольники в графическом контексте.

## Решение

Воспользуйтесь `CGPathAddRect` для добавления прямоугольника к пути, а потом отрисовывайте этот путь в графическом контексте.

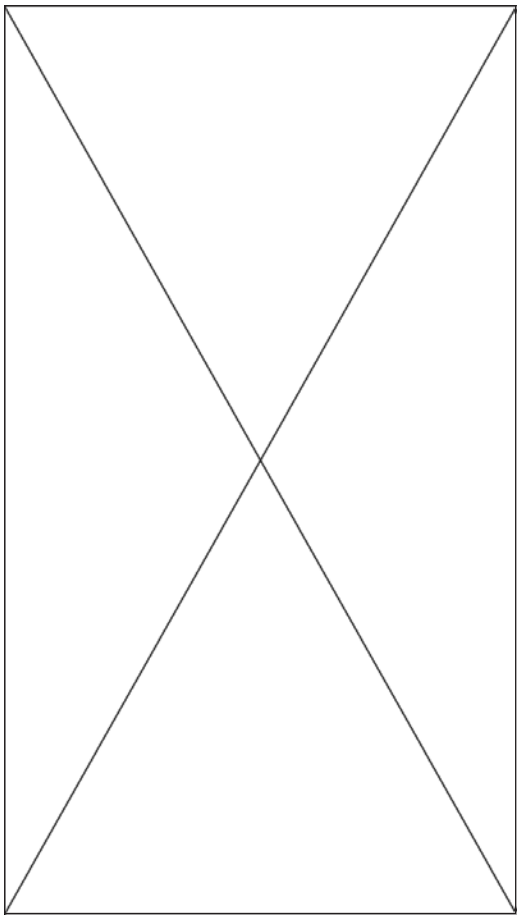

**Рис. 17.21.** Рисование в графическом контексте с использованием путей

## Обсуждение

Как мы узнали из раздела 17.7, создавать и использовать пути довольно просто. Одна из процедур, которую Core Graphics позволяет использовать с путями, — `CGPathAddRect`. Она позволяет отрисовывать прямоугольники как части путей. Вот пример:

```
- (void)drawRect:(CGRect)rect{

  /* Сначала создаем путь. Просто описатель пути. */
  CGMutablePathRef path = CGPathCreateMutable();

  /* Это границы прямоугольника. */
  CGRect rectangle = CGRectMake(10.0f,
                                10.0f,
                                200.0f,
```

```
                                300.0f);

  /* Добавляем прямоугольник к пути. */
  CGPathAddRect(path,
                NULL,
                rectangle);

  /* Получаем описатель текущего контекста. */
  CGContextRef currentContext = UIGraphicsGetCurrentContext();

  /* Добавляем путь к контексту. */
  CGContextAddPath(currentContext,
                   path);

  /* Задаем голубой в качестве цвета заливки. */
  [[UIColor colorWithRed:0.20f
                   green:0.60f
                    blue:0.80f
                   alpha:1.0f] setFill];

  /* Задаем для обводки коричневый цвет. */
  [[UIColor brownColor] setStroke];

  /* Задаем для ширины (обводки) значение 5. */
  CGContextSetLineWidth(currentContext,
                        5.0f);

  /* Проводим путь в контексте и применяем к нему заливку. */
  CGContextDrawPath(currentContext,
                    kCGPathFillStroke);

  /* Избавляемся от пути. */
  CGPathRelease(path);

}
```

Здесь мы рисуем на пути прямоугольник, который впоследствии заполняем голубым цветом, а края прямоугольника отрисовываем коричневым. На рис. 17.22 показано, что мы увидим, запустив эту программу (конечно же, цвета на черно-белой иллюстрации не видны).

Если вы собираетесь отрисовать несколько прямоугольников, то можете передать массив объектов `CGRect` процедуре `CGPathAddRects`. Вот пример:

```
- (void)drawRect:(CGRect)rect{

  /* Сначала создаем путь. Просто описатель пути. */
  CGMutablePathRef path = CGPathCreateMutable();

  /* Это границы первого прямоугольника. */
  CGRect rectangle1 = CGRectMake(10.0f,
                                 10.0f,
```

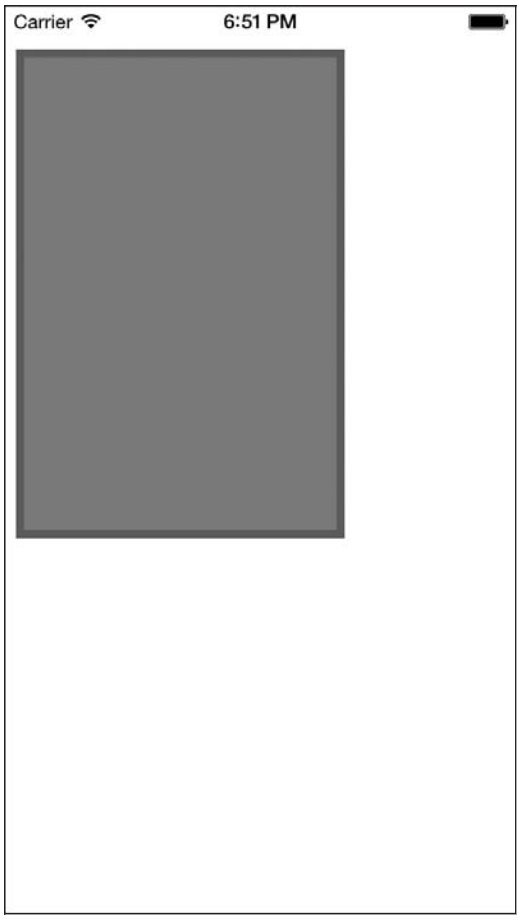


**Рис. 17.22.** Отрисовка прямоугольника с помощью путей

```
                                200.0f,
                                300.0f);

/* Это границы второго прямоугольника. */
CGRect rectangle2 = CGRectMake(40.0f,
                                100.0f,
                                90.0f,
                                300.0f);

/* Помещаем оба прямоугольника в массив. */
CGRect rectangles[2] = {
  rectangle1, rectangle2
};

/* Добавляем прямоугольники к пути. */
CGPathAddRects(path,
```

```
                NULL,
                (const CGRect *)&rectangles,
                2);

  /* Получаем описатель текущего контекста. */
  CGContextRef currentContext = UIGraphicsGetCurrentContext();

  /* Добавляем путь к контексту. */
  CGContextAddPath(currentContext,
                   path);

  /* Задаем голубой в качестве цвета заливки. */
  [[UIColor colorWithRed:0.20f
                   green:0.60f
                    blue:0.80f
                   alpha:1.0f] setFill];

  /* Задаем для обводки черный цвет. */
  [[UIColor blackColor] setStroke];

  /* Задаем для ширины (обводки) значение. 5 */
  CGContextSetLineWidth(currentContext,
                        5.0f);

  /* Проводим путь в контексте и применяем к нему заливку. */
  CGContextDrawPath(currentContext,
                    kCGPathFillStroke);

  /* Избавляемся от пути. */
  CGPathRelease(path);

}
```

На рис. 17.23 показано, как результат выполнения этого кода будет выглядеть в симуляторе iOS. Мы передаем процедуре `CGPathAddRects` следующие параметры (именно в таком порядке).

1. Описатель пути, к которому мы будем добавлять прямоугольники.
2. Преобразование (при его наличии), которое потребуется применить к прямоугольникам. (Подробнее о преобразованиях рассказано в разделах 17.11–17.13.)
3. Ссылку на массив `CGRect`, в котором содержатся прямоугольники.
4. Количество прямоугольников в массиве, который мы передали в предыдущем параметре. Исключительно важно передать именно столько прямоугольников, сколько содержится в вашем массиве, чтобы избежать непредвиденного поведения этой процедуры.

## См. также

Разделы 17.7, 17.11–17.13.

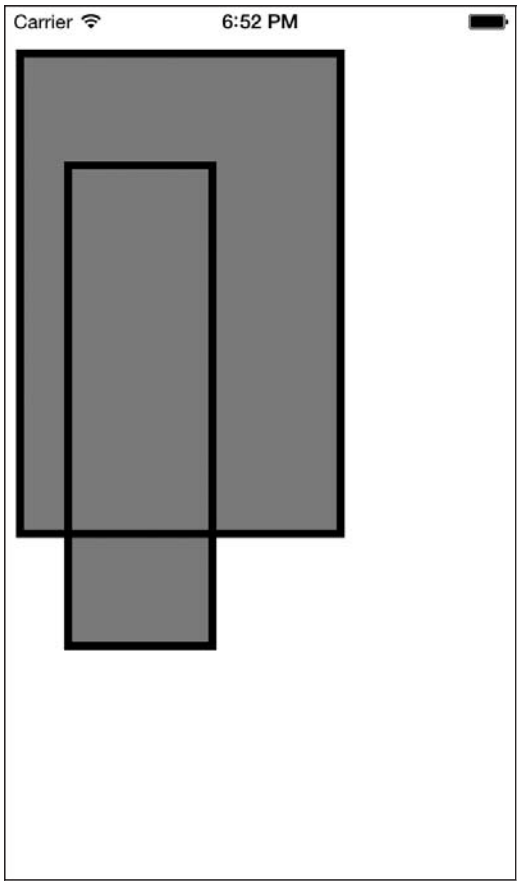


**Рис. 17.23.** Одновременная отрисовка нескольких прямоугольников

# 17.9. Добавление теней к фигурам

## Постановка задачи

Требуется применять тени к тем фигурам, которые вы отрисовываете в графическом контексте.

## Решение

Воспользуйтесь процедурой `CGContextSetShadow`.

## Обсуждение

В Core Graphics рисовать тени не составляет никакого труда. Графический контекст — это и есть элемент, несущий на себе тень. Это означает, что от вас требуется

просто применить тень к контексту, отрисовать для нее необходимые контуры, а потом удалить тень с контекста (или задать новый контекст). Чуть позже мы рассмотрим эти операции на примере.

В Core Graphics для применения тени к графическому контексту могут использоваться две процедуры:

- `CGContextSetShadow` — создает черные или серые тени, принимает три параметра:
  - графический контекст, к которому следует применить тень;
  - отступ, указываемый значением типа `CGSize`, на который тень распространяется вправо и вниз от каждой фигуры. Чем больше значение *x* данного отступа, тем больше тень будет распространяться вправо. Чем больше значение *y*, тем ниже будет тень;
  - значение размытия, которое следует применить к тени, указывается как число с плавающей точкой (`CGFloat`). Если задать для данного параметра значение `0.0f`, то у тени будут абсолютно четкие контуры. Чем выше это значение, тем более размытой будет становиться тень. Далее будет приведен соответствующий пример;
- `CGContextSetShadowWithColor` — принимает такие же параметры, как и `CGContextSetShadow`, плюс еще один. Этот четвертый параметр типа `CGColorRef` задает цвет тени.

В начале этого подраздела я отмечал, что графический контекст сохраняет свойства расположенных в нем теней, пока мы специально не удалим тень. Хотелось бы дополнительно разъяснить этот момент на примере. Напишем код, позволяющий нам отрисовать два прямоугольника: первый с тенью, второй — без нее. Первый прямоугольник нарисуем так:

```
- (void) drawRectAtTopOfScreen{

  /* Получаем описатель для актуального контекста. */
  CGContextRef currentContext = UIGraphicsGetCurrentContext();

  CGContextSetShadowWithColor(currentContext,
                              CGSizeMake(10.0f, 10.0f),
                              20.0f,
                              [[UIColor grayColor] CGColor]);

  /* Сначала создаем путь. Просто описатель пути. */
  CGMutablePathRef path = CGPathCreateMutable();

  /* Это границы прямоугольника. */
  CGRect firstRect = CGRectMake(55.0f,
                                60.0f,
                                150.0f,
                                150.0f);

  /* Добавляем прямоугольник к пути. */
  CGPathAddRect(path,
```

```
                NULL,
                firstRect);

  /* Добавляем путь к контексту. */
  CGContextAddPath(currentContext,
                   path);

  /* Задаем голубой в качестве цвета заливки. */
  [[UIColor colorWithRed:0.20f
                   green:0.60f
                    blue:0.80f
                   alpha:1.0f] setFill];

  /* Заполняем путь в контексте цветом заливки. */
  CGContextDrawPath(currentContext,
                    kCGPathFill);

  /* Избавляемся от пути. */
  CGPathRelease(path);

}
- (void) drawRect:(CGRect)rect{

[self drawRectAtTopOfScreen];
}
```

Если вызвать этот метод в методе экземпляра `drawRect:` объекта-вида, то на экране появится прямоугольник с красивой тенью, как мы и хотели (рис. 17.24).

Теперь нарисуем второй прямоугольник. Мы не будем специально запрашивать тень, а оставим свойство тени графического контекста таким же, как и в первом прямоугольнике:

```
- (void) drawRectAtBottomOfScreen{

  /* Получаем описатель текущего контекста. */
  CGContextRef currentContext = UIGraphicsGetCurrentContext();

  CGMutablePathRef secondPath = CGPathCreateMutable();

  CGRect secondRect = CGRectMake(150.0f,
                                 250.0f,
                                 100.0f,
                                 100.0f);

  CGPathAddRect(secondPath,
                NULL,
                secondRect);

  CGContextAddPath(currentContext,
                   secondPath);
  [[UIColor purpleColor] setFill];
```

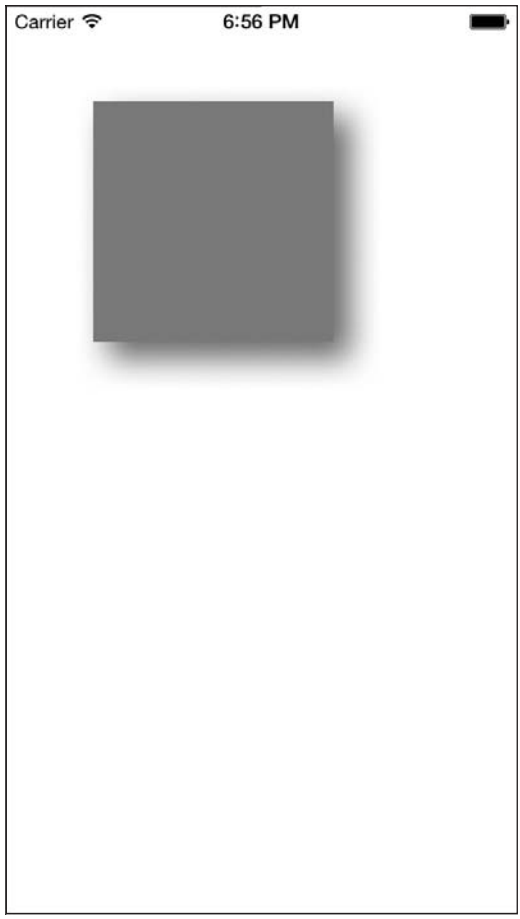


**Рис. 17.24.** Тень, примененная к прямоугольнику

```
  CGContextDrawPath(currentContext,
                    kCGPathFill);

  CGPathRelease(secondPath);

}

- (void)drawRect:(CGRect)rect{
  [self drawRectAtTopOfScreen];
  [self drawRectAtBottomOfScreen];
}
```

Метод `drawRect:` сначала вызывает метод `drawRectAtTopOfScreen`, а сразу же после этого — метод `drawRectAtBottomOfScreen`. Мы не запрашивали создание тени для прямоугольника `drawRectAtBottomOfScreen`, но после запуска кода вы увидите примерно такой результат, как на рис. 17.25.

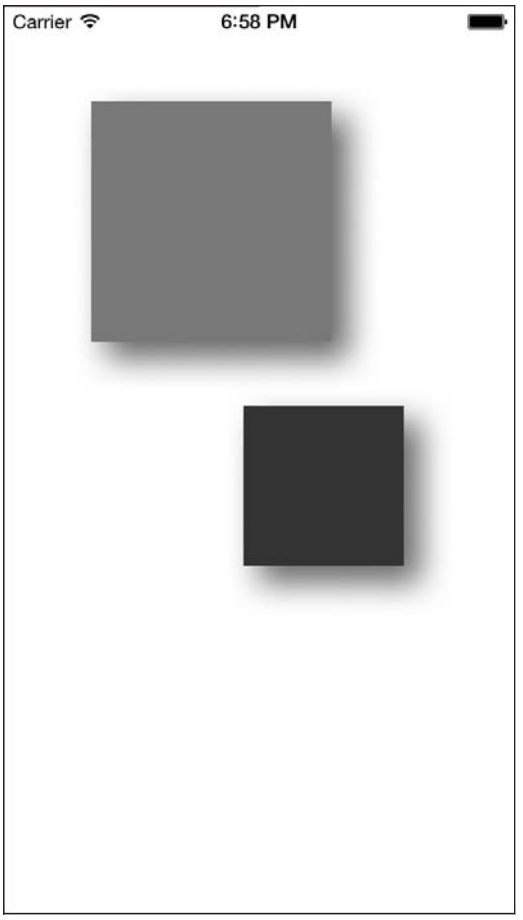


**Рис. 17.25.** Мы не собирались применять тень ко второму прямоугольнику, но она есть

Сразу заметно, что тень применена и ко второму прямоугольнику, расположенному в нижней части экрана. Чтобы избежать этого, мы сохраним графический контекст еще до применения к нему тени, а потом, когда захотим удалить теневой эффект, восстановим это состояние.

В широком смысле прием сохранения и последующего восстановления графического контекста работает не только с тенями. В ходе такой операции восстанавливаются все данные, связанные с графическим контекстом (цвет заливки, шрифт, толщина линий и т. д.), — они возвращаются к установленным ранее значениям. Так, например, если до восстановления графического контекста вы работали с иными цветами заливки и обводки, чем те, что заданы в нем, то эти цвета будут сброшены.

Можно сохранять состояние графического контекста с помощью процедуры `CGContextSaveGState` и восстанавливать его прежнее состояние, используя процедуру `CGContextRestoreGState`. Так, если мы изменим процедуру `drawRectAtTopOfScreen`, сохранив состояние графического контекста до применения тени, а потом восста-

новим это состояние после того, как отрисуем путь, то результаты у нас получатся иные (рис. 17.26):

```
- (void) drawRectAtTopOfScreen{

  /* Получаем описатель текущего контекста. */
  CGContextRef currentContext = UIGraphicsGetCurrentContext();

  CGContextSaveGState(currentContext);

  CGContextSetShadowWithColor(currentContext,
                              CGSizeMake(10.0f, 10.0f),
                              20.0f,
                              [[UIColor grayColor] CGColor]);

  /* Сначала создаем путь. Просто описатель пути. */
  CGMutablePathRef path = CGPathCreateMutable();

  /* Это границы прямоугольника. */
  CGRect firstRect = CGRectMake(55.0f,
                                60.0f,
                                150.0f,
                                150.0f);

  /* Добавляем прямоугольник к пути. */
  CGPathAddRect(path,
                NULL,
                firstRect);

  /* Добавляем путь к контексту. */
  CGContextAddPath(currentContext,
                   path);

  /* Задаем голубой в качестве цвета заливки. */
  [[UIColor colorWithRed:0.20f
                   green:0.60f
                    blue:0.80f
                   alpha:1.0f] setFill];

  /* Проводим путь в контексте и применяем к нему заливку. */
  CGContextDrawPath(currentContext,
                    kCGPathFill);

  /* Избавляемся от пути. */
  CGPathRelease(path);

  /* Восстанавливаем контекст в исходном состоянии
     (в котором мы начали с ним работать). */
  CGContextRestoreGState(currentContext);

}
```

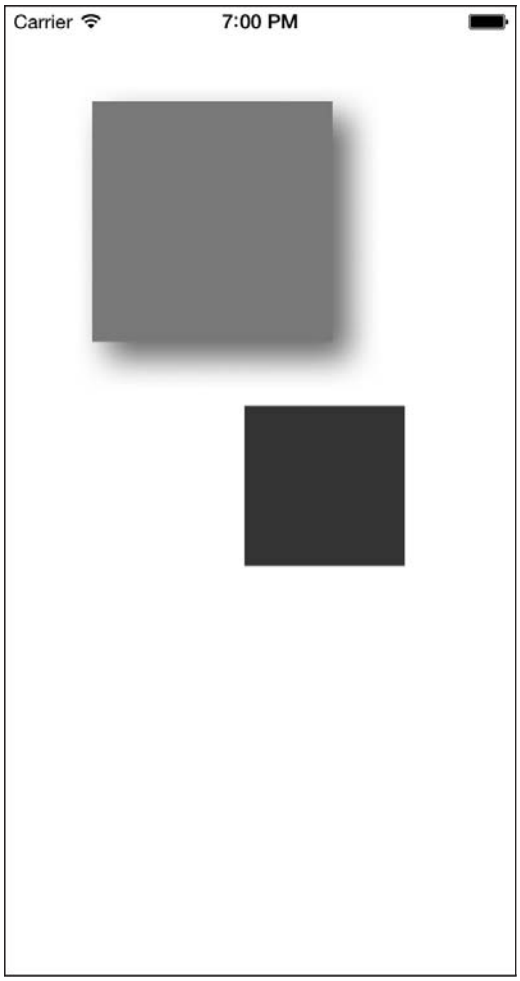


**Рис. 17.26.** Сохранение состояния графического контекста для точного отображения теней

# 17.10. Отрисовка градиентов

## Постановка задачи

Требуется рисовать в графическом контексте градиенты, используя различные цвета.

## Решение

Воспользуйтесь функцией `CGGradientCreateWithColor`.

## Обсуждение

Мы уже поговорили о цвете в разделе 17.3 и теперь попробуем воспользоваться нашими навыками для решения более интересных задач, чем рисование простых прямоугольников и разноцветного текста.

В Core Graphics программист может создавать градиенты двух типов: осевой и радиальный (но мы обсудим только осевые градиенты). Осевой градиент начинается в определенной точке с одного цвета и заканчивается в другой точке иным цветом (конечно, градиент можно и начать и закончить одним и тем же цветом, но тогда он будет не слишком напоминать градиент). «Осевой» означает «относящийся к оси». Между двумя точками (начальной и конечной) создается сегмент линии, он и будет той осью, вдоль которой отрисовывается градиент. Образец осевого градиента показан на рис. 17.27. На самом деле это осевой градиент, начинающийся с голубого цвета и заканчивающийся зеленым, но на черно-белой иллюстрации этого, конечно же, не видно.


**Рис. 17.27.** Осевой градиент

Чтобы создать осевой градиент, вызовите функцию `CGGradientCreateWithColorComponents`. Возвращаемым значением этой функции будет новый градиент типа `CGGradientRef`. Это описатель градиента. Закончив работать с градиентом, *необходимо* вызвать процедуру `CGGradientRelease`, передав описатель тому градиенту, который вы ранее получили от `CGGradientCreateWithColorComponents`.

Функция `CGGradientCreateWithColorComponents` принимает четыре параметра.

- *Цветовое пространство* — это контейнер для цветового диапазона, он должен относиться к типу `CGColorSpaceRef`. Для этого параметра можем просто передать возвращаемое значение функции `CGColorSpaceCreateDeviceRGB` — и получим пространство цветов RGB.
- *Массив цветовых компонентов* (подробнее об этом — в разделе 17.3) — здесь должны содержаться значения красного, зеленого, голубого цветов и альфа-значение, все они относятся к типу `CGFloat`. Количество элементов в массиве тесно связано со значениями следующих двух параметров. Вы должны будете включить в этот массив такое количество значений, которого будет достаточно для того, чтобы указать ряд положений, обозначенных в четвертом параметре. Так, если вы запрашиваете только два положения (начальную и конечную точки), то в этом массиве должно быть минимум два цвета. А поскольку в состав каждого цвета входят красный, зеленый, голубой компоненты, а также альфа-значение, в данном массиве должно быть $2 \times 4$ элемента: четыре для первого цвета и четыре — для второго. Не волнуйтесь, если пока не все понимаете, — все встанет на свои места после изучения примеров.
- *Положения оттенков в цветовом массиве* — этот параметр определяет, как быстро в градиенте осуществляется переход от одного оттенка к другому. Количество элементов должно быть таким же, как и значение четвертого параметра. Например, если вы запрашиваете четыре цвета и хотите, чтобы первый цвет был начальным цветом градиента, а последний цвет располагался в конце градиента, то нужно предоставить массив из двух элементов типа `CGFloat`, где первый элемент имеет значение `0.0f` (как в *первом* компоненте цветового массива), а второй элемент — `3.0f` (как в *четвертом* компоненте цветового массива). Значения двух промежуточных цветов определяют, в каком именно порядке расположены в градиенте оттенки, лежащие между начальным и конечным цветами. Опять же не волнуйтесь, если это сложно сразу усвоить. Я приведу много примеров, на которых вся концепция станет совершенно ясна.
- *Количество положений* — здесь мы указываем, сколько цветов и положений должно быть в градиенте.

Рассмотрим пример. Предположим, мы хотим нарисовать градиент, который показан на рис. 17.27. Вот как это делается.

1. Выбираем начальную и конечную точки градиента — ось, вдоль которой будут изменяться оттенки. В данном случае я указываю переход слева направо. Представьте, что цвет изменяется по мере движения вдоль гипотетической линии. Цвета будут располагаться по оси так, что любая вертикальная линия, перпендикулярно пересекающая ось градиента, будет пролегать только по одному оттенку. В случае, показанном на рис. 17.27, любая вертикальная линия будет

перпендикулярна оси градиента. Рассмотрим эти вертикальные линии подробнее. Действительно, в любой ее точке цвет градиента один и тот же. Вот так и строится градиент. Хорошо, хватит теории — переходим ко второму этапу.

2. Теперь нам нужно создать цветовое пространство, которое будет передано функции `CGGradientCreateWithColorComponents` в первом параметре, как было объяснено ранее:

```
CGColorSpaceRef colorSpace =
CGColorSpaceCreateDeviceRGB();
```

Закончив работу с этим цветовым пространством, мы избавимся от него.

3. Зададим голубой в качестве начального цвета (слева), а зеленый — в качестве конечного (справа), как показано на рис. 17.27. Названия, которыми я пользуюсь (`startColorComponents` и `endColorComponents`), выбраны произвольно и помогают нам не забыть о положении каждого цвета. Для указания того, какой цвет будет находиться в начале, а какой — в конце, мы воспользуемся позициями из массива:

```
UIColor *startColor = [UIColor blueColor];
  CGFloat *startColorComponents =
  (CGFloat *)CGColorGetComponents([startColor CGColor]);

  UIColor *endColor = [UIColor greenColor];
  CGFloat *endColorComponents =
  (CGFloat *)CGColorGetComponents([endColor CGColor]);
```

Если вы забыли, какая концепция лежит в основе цветовых компонентов, вернитесь к этому вопросу, изложенному в разделе 17.3, а потом продолжайте читать.

4. Получив компоненты каждого цвета, мы помещаем все их в одномерный массив, который будет передан функции `CGGradientCreateWithColorComponents`:

```
CGFloat colorComponents[8] = {

  /* Четыре компонента оранжевого цвета (RGBA) */
  startColorComponents[0],
  startColorComponents[1],
  startColorComponents[2],
  startColorComponents[3], /* Первый цвет = оранжевый */

  /* Четыре компонента голубого цвета (RGBA) */
  endColorComponents[0],
  endColorComponents[1],
  endColorComponents[2],
  endColorComponents[3], /* Второй цвет = голубой */

};
```

5. Поскольку у нас в этом массиве всего два цвета, следует указать, что первый цвет расположен в самом начале градиента (точка с координатами (0; 0)) а второй — в самом конце (точка (1; 0)). Итак, поместим эти показатели в массив, предназначенный для передачи функции CGGradientCreateWithColorComponents:

```
CGFloat colorIndices[2] = {
  0.0f, /* Цвет 0 в массиве colorComponents */
  1.0f, /* Цвет 1 в массиве colorComponents */
};
```

6. Теперь нам остается просто вызвать функцию CGGradientCreateWithColorComponents со всеми сгенерированными значениями:

```
CGGradientRef gradient =
CGGradientCreateWithColorComponents
(colorSpace,
  (const CGFloat *)&colorComponents,
  (const CGFloat *)&colorIndices,
2);
```

7. Прекрасно! Теперь в переменной gradient находится объект градиента. Пока не забыли, нужно высвободить цветовое пространство, созданное с помощью функции CGColorSpaceCreateDeviceRGB:

```
CGColorSpaceRelease(colorSpace);
```

Теперь воспользуемся процедурой CGContextDrawLinearGradient для отрисовки осевого градиента в графическом контексте. Эта процедура принимает пять параметров.

- *Графический контекст* — указывает графический контекст, в котором будет отрисовываться осевой градиент.
- *Осевой градиент* — описатель объекта осевого градиента. Этот объект градиента создан с помощью функции CGGradientCreateWithColorComponents.
- *Начальная точка* — точка в графическом контексте, указанная в параметре CGPoint, в которой начинается градиент.
- *Конечная точка* — точка в графическом контексте, указанная в параметре CGPoint, в которой заканчивается градиент.
- *Параметры отрисовки градиента* — указывают, что должно произойти, если начальная и конечная точки не совпадают с краями графического контекста. Для заполнения пространства, лежащего вне градиента, можно использовать начальный или конечный цвета. Этот параметр может принимать одно из следующих значений:
  - kCGGradientDrawsAfterEndLocation — распространяет градиент на все точки после конечной точки градиента;
  - kCGGradientDrawsBeforeStartLocation — распространяет градиент на все точки до начальной точки градиента;
  - 0 — градиент не распространяется.

Чтобы распространить градиент в обе стороны, укажите оба параметра — «до» и «после», — воспользовавшись логическим оператором ИЛИ (обозначается символом |). Пример будет рассмотрен далее:

```
CGRect screenBounds = [[UIScreen mainScreen] bounds];

CGPoint startPoint, endPoint;

startPoint = CGPointMake(0.0f,
                         screenBounds.size.height / 2.0f);

endPoint = CGPointMake(screenBounds.size.width,
                       startPoint.y);

CGContextDrawLinearGradient
(currentContext,
 gradient,
 startPoint,
 endPoint,
 0);

CGGradientRelease(gradient);
```

Описатель градиента, который мы высвобождаем в конце этого кода, был создан в другом блоке кода в одном из предыдущих примеров.

Очевидно, что результат выполнения этого кода будет напоминать рис. 17.27. Поскольку мы начали градиент с самой левой точки экрана и распространили его до самой правой, то не можем воспользоваться теми значениями, которые способен получить последний параметр процедуры `CGContextDrawLinearGradient`, *параметр отрисовки градиента*. Исправим этот недостаток. Попробуем нарисовать такой градиент, как на рис. 17.28.

При написании кода воспользуемся той же процедурой, о которой говорили ранее:

```
- (void)drawRect:(CGRect)rect{

  CGContextRef currentContext = UIGraphicsGetCurrentContext();

  CGContextSaveGState(currentContext);
  CGColorSpaceRef colorSpace =
  CGColorSpaceCreateDeviceRGB();

  UIColor *startColor = [UIColor orangeColor];
  CGFloat *startColorComponents =
  (CGFloat *)CGColorGetComponents([startColor CGColor]);

  UIColor *endColor = [UIColor blueColor];
  CGFloat *endColorComponents =
```

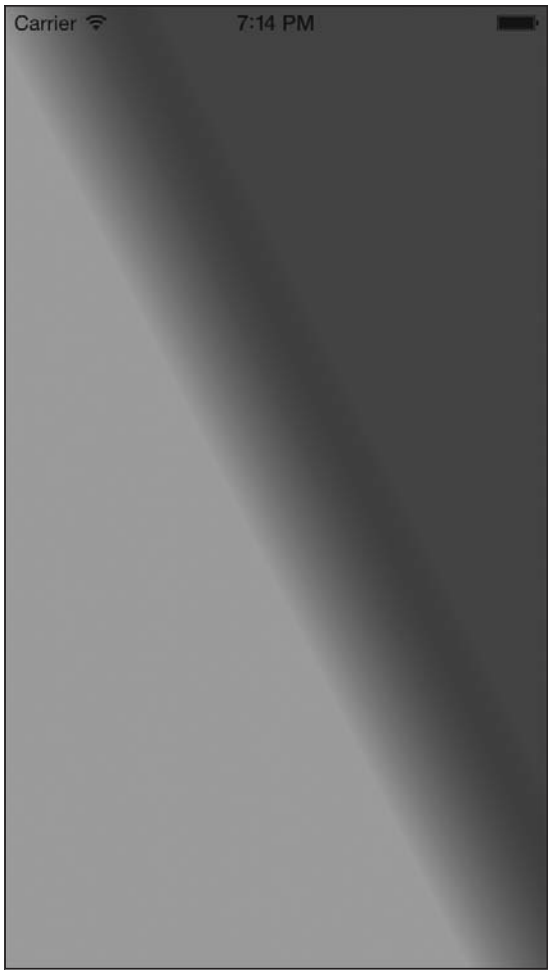


**Рис. 17.28.** Осевой градиент с оттенками, распространяющимися за его начальную и конечную точки

```
(CGFloat *)CGColorGetComponents([endColor CGColor]);

CGFloat colorComponents[8] = {

  /* Четыре компонента оранжевого цвета (RGBA (RGBA) */
  startColorComponents[0],
  startColorComponents[1],
  startColorComponents[2],
  startColorComponents[3], /* Первый цвет = оранжевый */

  /* Четыре компонента голубого цвета (RGBA) */
  endColorComponents[0],
  endColorComponents[1],
```

```
    endColorComponents[2],
    endColorComponents[3], /* Второй цвет = голубой */

  };

  CGFloat colorIndices[2] = {
    0.0f, /* Цвет 0 в массиве colorComponents */
    1.0f, /* Цвет 1 в массиве colorComponents */
  };

  CGGradientRef gradient = CGGradientCreateWithColorComponents
  (colorSpace,
   (const CGFloat *)&colorComponents,
   (const CGFloat *)&colorIndices,
   2);

  CGColorSpaceRelease(colorSpace);

  CGPoint startPoint, endPoint;

  startPoint = CGPointMake(120,
                           260);
  endPoint = CGPointMake(200.0f,
                         220);
  CGContextDrawLinearGradient (currentContext,
                               gradient,
                               startPoint,
                               endPoint,
                               kCGGradientDrawsBeforeStartLocation |
                               kCGGradientDrawsAfterEndLocation);
  CGGradientRelease(gradient);
  CGContextRestoreGState(currentContext);
}
```

Возможно, вам не совсем понятно, как при смешивании значений `kCGGradientDrawsBeforeStartLocation` и `kCGGradientDrawsAfterEndLocation`, переданных процедуре `CGContextDrawLinearGradient`, получается диагональный эффект, как на рис. 17.28. Поэтому уберем эти значения и зададим для этого параметра процедуры `CGContextDrawLinearGradient` значение `0` — как и раньше. Результат получится как на рис. 17.29.

На рис. 17.28 и 17.29 изображены одинаковые градиенты. Но у градиента на рис. 17.28 цвета начальной и конечной точек распространяются по обе стороны градиента на весь графический контекст, поэтому весь экран оказывается закрашен.

## См. также

Раздел 17.3.

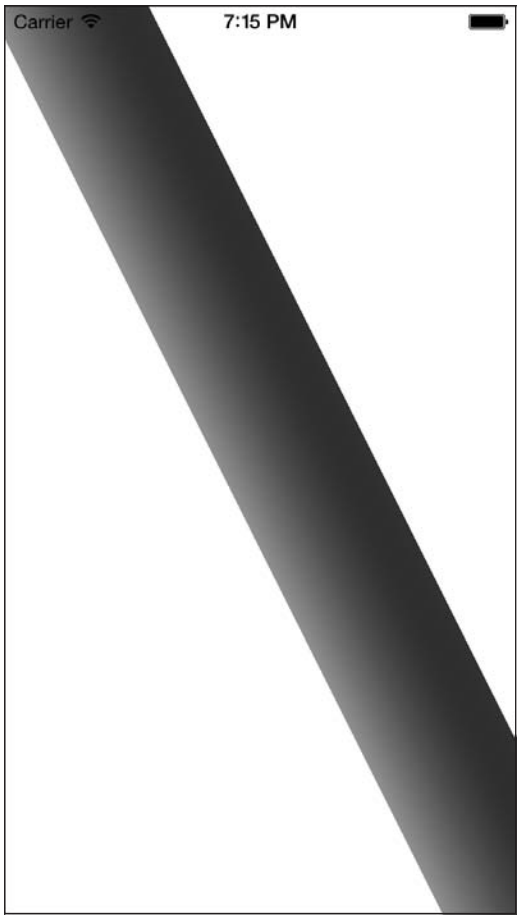


**Рис. 17.29.** Осевой градиент без распространения цветов

# 17.11. Перемещение фигур, нарисованных в графических контекстах

## Постановка задачи

Требуется переместить все, что изображено в графическом контексте, на новое место, не изменяя при этом кода отрисовки, либо просто без труда сместить содержимое графического контекста.

## Решение

Воспользуйтесь функцией `CGAffineTransformMakeTranslation` для создания аффинного преобразования сдвига.

## Обсуждение

В разделе 17.8 было упомянуто о преобразованиях. Преобразование — это, в сущности, просто изменение способа отображения рисунка. Преобразования в Core Graphics — это объекты, применяемые к фигурам перед отрисовкой последних. Например, можно создать преобразование сдвига (Translation Transformation). «Сдвига чего?» — могли бы спросить вы. Дело в том, что преобразование сдвига — это механизм, позволяющий *сместить* фигуру или графический контекст.

Среди других типов преобразований следует также назвать вращение (см. раздел 17.13) и масштабирование (см. раздел 17.12). Все это примеры *аффинных* преобразований, то есть при таком преобразовании каждая точка оригинала сопоставляется с другой точкой в окончательной версии. Все преобразования, о которых мы будем говорить в этой книге, являются аффинными.

В ходе преобразования сдвига актуальное положение фигуры на пути или в графическом контексте сдвигается на другую относительную позицию. Например, если вы поставите точку с координатами (10; 20), примените к ней преобразование сдвига (30; 40) и снова ее поставите, точка окажется расположенной в координатах (40; 60), поскольку 40 = 10 + 30, а 60 = 20 + 40.

Чтобы создать новое преобразование сдвига, используется функция `CGAffineTransformMakeTranslation`, которая возвращает аффинное преобразование типа `CGAffineTransform`. Два параметра этой функции указывают сдвиг по осям *X* и *Y* в точках.

В разделе 17.8 мы изучили, что процедура `CGPathAddRect` принимает в качестве второго параметра объект преобразования типа `CGAffineTransform`. Чтобы сместить прямоугольник с его исходной позиции на другую, можно просто создать аффинное преобразование, указывающее изменения, которые вы хотели бы применить к координатам *x* и *y*, и передать преобразование второму параметру процедуры `CGPathAddRect`, как показано далее:

```
- (void)drawRect:(CGRect)rect{

  /* Сначала создаем путь. Просто описатель пути. */
  CGMutablePathRef path = CGPathCreateMutable();

  /* Это границы прямоугольника. */
  CGRect rectangle = CGRectMake(10.0f,
                                10.0f,
                                200.0f,
                                300.0f);

  /* Мы хотим сместить прямоугольник на 100 точек вправо,
     не изменив при этом его положения по оси Y. */
  CGAffineTransform transform = CGAffineTransformMakeTranslation(100.0f,
                                                                 0.0f);

  /* Добавляем прямоугольник к пути. */
  CGPathAddRect(path,
                &transform,
```

```
                rectangle);

  /* Получаем описатель текущего контекста. */
  CGContextRef currentContext =
  UIGraphicsGetCurrentContext();

  /* Добавляем путь к контексту. */
  CGContextAddPath(currentContext,
                   path);

  /* Задаем голубой в качестве цвета заливки. */
  [[UIColor colorWithRed:0.20f
                   green:0.60f
                    blue:0.80f
                   alpha:1.0f] setFill];

  /* Задаем для обводки коричневый цвет. */
  [[UIColor brownColor] setStroke];

  /* Задаем для ширины (обводки) значение 5. */
  CGContextSetLineWidth(currentContext,
                        5.0f);

  /* Проводим путь в контексте и применяем к нему заливку. */
  CGContextDrawPath(currentContext,
                    kCGPathFillStroke);

  /* Избавляемся от пути. */
  CGPathRelease(path);

}
```

На рис. 17.30 показан результат выполнения этого блока кода внутри объекта-вида.

Сравните рис. 17.30 и 17.22. Видите разницу? Еще раз просмотрите исходный код для обеих фигур и убедитесь в том, что положения по осям *X* и *Y*, указанные для обоих прямоугольников, в обоих блоках кода идентичны. Различие заключается только в том, что на рис. 17.30 мы видим результат применения к прямоугольнику аффинного преобразования, когда прямоугольник добавляется к пути.

Кроме применения преобразований к фигурам, отрисовываемым относительно путей, мы можем применять преобразования и к графическому контексту с помощью процедуры `CGContextTranslateCTM`. Она применяет преобразование к текущей матрице преобразований (Current Transformation Matrix, CTM). Хотя это название и может показаться сложным, понять его смысл не составляет труда. Считайте CTM правилами, определяющими расположение центра вашего графического контекста, а также правилами проецирования каждой отрисовываемой точки на экране. Например, если вы приказываете Core Graphics поставить точку с координатами (0; 0), Core Graphics найдет центр экрана, получив эту информацию из текущей матрицы преобразований. Затем CTM вы-

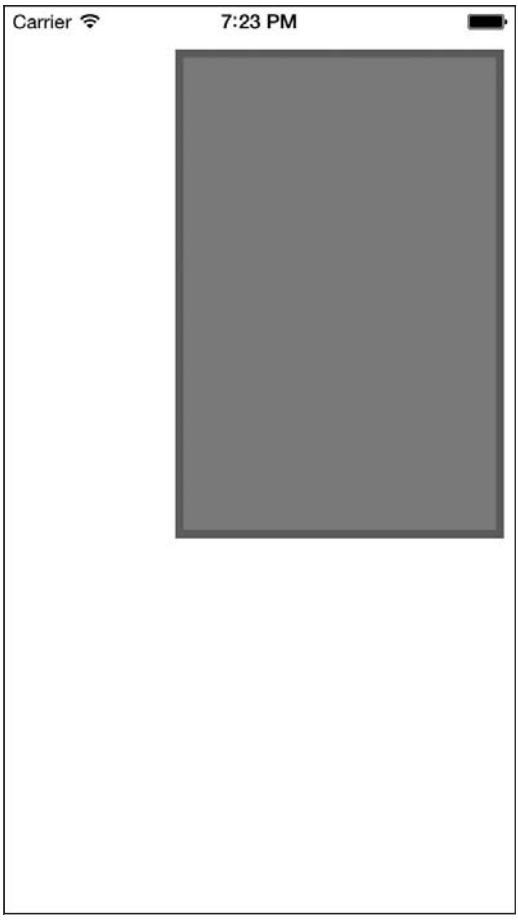


**Рис. 17.30.** Прямоугольник с аффинным преобразованием сдвига

полнит определенные вычисления и сообщит Core Graphics, что искомая точка расположена в верхнем левом углу экрана. С помощью таких процедур, как `CGContextTranslateCTM`, можно изменить конфигурацию этой матрицы, после чего заставить все фигуры, отрисованные в графическом контексте, занять на холсте другие позиции. Вот пример, в котором мы достигаем точно такого же эффекта, как и на рис. 17.30, но применяем преобразование сдвига не к самому прямоугольнику, а к текущей матрице преобразований:

```
- (void)drawRect:(CGRect)rect{

  /* Сначала создаем путь. Просто описатель пути. */
  CGMutablePathRef path = CGPathCreateMutable();

  /* Это границы прямоугольника. */
  CGRect rectangle = CGRectMake(10.0f,
                                10.0f,
```

```
                                200.0f,
                                300.0f);

  /* Добавляем прямоугольник к пути. */
  CGPathAddRect(path,
                NULL,
                rectangle);

  /* Получаем описатель текущего контекста. */
  CGContextRef currentContext = UIGraphicsGetCurrentContext();

  /* Сохраняем состояние контекста, чтобы позже
     можно было восстановить это состояние. */
  CGContextSaveGState(currentContext);

  /* Сдвигаем текущую матрицу преобразований
     на 100 точек вправо. */
  CGContextTranslateCTM(currentContext,
                        100.0f,
                        0.0f);

  /* Добавляем путь к контексту. */
  CGContextAddPath(currentContext,
                   path);

  /* Задаем голубой в качестве цвета заливки. */
  [[UIColor colorWithRed:0.20f
                   green:0.60f
                    blue:0.80f
                   alpha:1.0f] setFill];

  /* Задаем для обводки коричневый цвет. */
  [[UIColor brownColor] setStroke];

  /* Задаем для ширины (обводки) значение 5. */
  CGContextSetLineWidth(currentContext,
                        5.0f);

  /* Проводим путь в контексте и применяем к нему заливку. */
  CGContextDrawPath(currentContext,
                    kCGPathFillStroke);

  /* Избавляемся от пути. */
  CGPathRelease(path);

  /* Восстанавливаем состояние контекста. */
  CGContextRestoreGState(currentContext);

}
```

Запустив эту программу, вы получите точно такие же результаты, как и на рис. 17.30.

## См. также

Разделы 17.8, 17.12 и 17.13.

# 17.12. Масштабирование фигур, нарисованных в графических контекстах

## Постановка задачи

Требуется динамически масштабировать фигуры, отрисованные в графическом контексте, в сторону уменьшения или увеличения.

## Решение

Создайте аффинное преобразование масштаба, воспользовавшись функцией `CGAffineTransformMakeScale`.

## Обсуждение

В разделе 17.11 было объяснено, что такое преобразование и как применять его к фигурам и графическим контекстам. Один из вариантов преобразования, которым вы можете воспользоваться, — это масштабирование. Core Graphics позволяет без проблем масштабировать фигуру, например круг, на 100 % относительно ее исходного размера.

Чтобы создать аффинное преобразование масштаба, пользуйтесь функцией `CGAffineTransformMakeScale`, которая возвращает объект преобразования типа `CGAffineTransform`. Если вы хотите применить преобразование масштаба непосредственно к графическому контексту, примените процедуру `CGContextScaleCTM`, которая масштабирует CTM. Подробнее о CTM (текущей матрице преобразований) рассказано в разделе 17.11.

Функции преобразования масштаба принимают два параметра: масштабирование по оси *X* и масштабирование по оси *Y*. Еще раз обратимся к прямоугольнику с рис. 17.22. Если мы хотим масштабировать этот прямоугольник, чтобы его исходные длина и ширина уменьшились вполовину, то можно просто масштабировать по оси *X* и *Y* на 0,5 (вполовину от исходного значения), как показано здесь:

```
/* Масштабируем прямоугольник, уменьшая его на половину. */
CGAffineTransform transform =
  CGAffineTransformMakeScale(0.5f, 0.5f);

/* Добавляем прямоугольник к пути. */
CGPathAddRect(path,
              &transform,
              rectangle);
```

На рис. 17.31 показано, что получится, когда мы применим преобразование масштаба к коду, написанному в разделе 17.8.

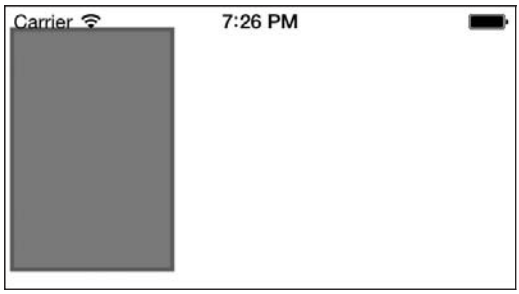


**Рис. 17.31.** Масштабирование прямоугольника

Дополнительно к функции `CGAffineTransformMakeScale` можно использовать процедуру `CGContextScaleCTM`, помогающую применить преобразование масштаба к графическому контексту. Следующий код даст тот же эффект, что и в предыдущем примере (вновь обратите внимание на рис. 17.31):

```
- (void)drawRect:(CGRect)rect{

  /* Сначала создаем путь. Просто описатель пути. */
  CGMutablePathRef path = CGPathCreateMutable();

  /* Это границы прямоугольника. */
  CGRect rectangle = CGRectMake(10.0f,
                                10.0f,
                                200.0f,
                                300.0f);

  /* Добавляем прямоугольник к пути. */
  CGPathAddRect(path,
                NULL,
                rectangle);

  /* Получаем описатель текущего контекста. */
  CGContextRef currentContext = UIGraphicsGetCurrentContext();

  /* Масштабируем все фигуры, отрисованные в графическом контексте,
     уменьшая их на половину. */
  CGContextScaleCTM(currentContext,
                    0.5f,
                    0.5f);

  /* Добавляем путь к контексту. */
  CGContextAddPath(currentContext,
                   path);

  /* Задаем голубой в качестве цвета заливки. */
  [[UIColor colorWithRed:0.20f
                   green:0.60f
                    blue:0.80f
                   alpha:1.0f] setFill];
```

```
    /* Задаем для обводки коричневый цвет. */
    [[UIColor brownColor] setStroke];

    /* Задаем для ширины (обводки) значение 5. */
    CGContextSetLineWidth(currentContext,
                          5.0f);

    /* Проводим путь в контексте и применяем к нему заливку. */
    CGContextDrawPath(currentContext,
                      kCGPathFillStroke);

    /* Избавляемся от пути. */
    CGPathRelease(path);

}
```

## См. также

Раздел 17.11.

# 17.13. Вращение фигур, нарисованных в графических контекстах

## Постановка задачи

Требуется иметь возможность вращать содержимое, отрисованное в графическом контексте, не изменяя при этом код отрисовки.

## Решение

Воспользуйтесь функцией `CGAffineTransformMakeRotation` для создания аффинного преобразования вращения.

## Обсуждение

Перед тем как приступать к работе с этим разделом, настоятельно рекомендую вам перечитать материал из разделов 17.11 и 17.12. Чтобы сократить текст, я старался не дублировать в последующих разделах материал, уже изложенный в предыдущих.

Точно так же, как при масштабировании и сдвиге, мы можем применять преобразование вращения к фигурам, отрисованным на путях и прямо в графическом контексте. Можно использовать функцию `CGAffineTransformMakeRotation` и сообщать значение вращения в радианах, а в качестве возвращаемого значения получать преобразование вращения типа `CGAffineTransform`. Затем это преобразование можно применять к путям и фигурам. Если вы хотите повернуть весь контекст на определенный угол, используйте процедуру `CGContextRotateCTM`.

Повернем прямоугольник, изображенный на рис. 17.22, на 45° по часовой стрелке (рис. 17.32). Значение, полученное в результате вращения, должно быть выражено в радианах. Положительные значения дают вращение по часовой стрелке, а отрицательные — против часовой:

```
/* Вращаем прямоугольник на 45° по часовой стрелке. */
CGAffineTransform transform =
CGAffineTransformMakeRotation((45.0f * M_PI) / 180.0f);

/* Добавляем прямоугольник к пути. */
CGPathAddRect(path,
              &transform,
              rectangle);
```

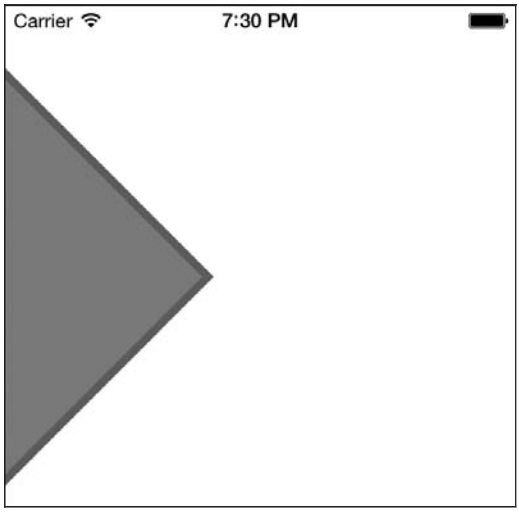


**Рис. 17.32.** Вращение прямоугольника

Как было показано в разделе 17.12, мы можем применить преобразование и непосредственно к графическому контексту — с помощью процедуры `CGContextRotateCTM`.

## См. также

Разделы 17.11 и 17.12.

# 17.14. Анимирование и перемещение видов

## Постановка задачи

Требуется анимировать смещение видов.

## Решение

При смещении видов используйте анимационные методы класса UIView.

## Обсуждение

В операционной системе iOS предоставляются различные способы выполнения анимации, среди этих возможностей есть как низкоуровневые, так и сравнительно высокоуровневые. Самый высокий уровень работы в данном случае обеспечивается во фреймворке UIKit, о котором мы также поговорим в этом разделе. В UIKit содержится некоторая низкоуровневая функциональность Core Animation, предоставляемая нам в форме довольно аккуратного API, с которым очень удобно работать.

Работа с анимацией в UIKit начинается с вызова метода класса beginAnimations:context:, относящегося к классу UIView. Первый параметр — это опциональное имя, которое вы можете выбрать для вашей анимации, а второй — опциональный контекст, который можно получить позже для передачи анимационным методам делегатов. Вскоре мы поговорим о них обоих.

После того как вы запустите анимацию с помощью метода beginAnimations:context:, она не начнет происходить, так как для этого потребуется еще вызвать метод класса commitAnimations, относящийся к классу UIView. Вычисления, которые вы производите над объектом-видом между вызовом beginAnimations:context: и commitAnimations (в результате которых этот вид, к примеру, перемещается), будут сопровождаться анимацией только после вызова commitAnimations. Рассмотрим пример.

Как упоминалось в разделе 17.4, я включил в пакет моего приложения рисунок Xcode.png. Это ярлык Xcode, который я нашел в картинках Google (см. рис. 17.9). Теперь в моем контроллере вида (см. введение к этой главе) я хочу поместить этот рисунок в виде с изображением типа UIImageView, а потом переместить этот вид с изображением из верхнего левого угла экрана в нижний правый угол.

Вот как мы решим эту задачу.

1. Откройте .h-файл вашего контроллера вида.
2. Определите экземпляр UIImageView как свойство контроллера вида и назовите его xcodeImageView:

```
#import "ViewController.h"
@interface ViewController ()

@property (nonatomic, strong) UIImageView *xcodeImageView;

@end
```

3. Когда вид загрузится, поместите изображение Xcode.png в экземпляр UIImage:

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

  UIImage *xcodeImage = [UIImage imageNamed:@"Xcode.png"];

  self.xcodeImageView = [[UIImageView alloc]
```

```
                                    initWithImage:xcodeImage];

    /* Просто задаем размеры, чтобы изображение уменьшилось. */
    [self.xcodeImageView setFrame:CGRectMake(0.0f,
                                             0.0f,
                                             100.0f,
                                             100.0f)];

    self.view.backgroundColor = [UIColor whiteColor];
    [self.view addSubview:self.xcodeImageView];

}
```

4. На рис. 17.33 показано, как будет выглядеть вид, когда программа запускается в симуляторе iOS.

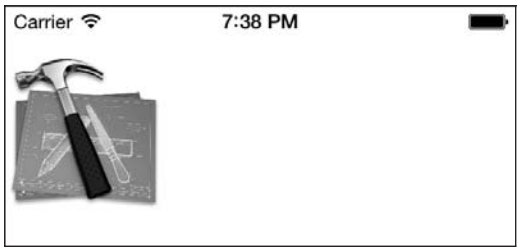


**Рис. 17.33.** Добавление вида с изображением в объект-вид

5. Теперь, когда вид появится на экране в методе экземпляра `viewDidAppear:` контроллера вида, приступим к исполнению анимационного блока, относящегося к виду с изображением. Эта анимация переместит изображение из исходной точки (в левом верхнем углу) в нижний правый угол. Кроме того, мы убедимся, что анимация произойдет за пятисекундный период:

```
- (void) viewDidAppear:(BOOL)paramAnimated{

    [super viewDidAppear:paramAnimated];

    /* Начинаем с верхнего левого угла. */
    [self.xcodeImageView setFrame:CGRectMake(0.0f,
                                             0.0f,
                                             100.0f,
                                             100.0f)];

    [UIView beginAnimations:@"xcodeImageViewAnimation"
                    context:(__bridge void *)self.xcodeImageView];

    /* Пятисекундная анимация. */
    [UIView setAnimationDuration:5.0f];

    /* Получаем делегаты анимации. */
```

```
    [UIView setAnimationDelegate:self];

    [UIView setAnimationDidStopSelector:
    @selector(imageViewDidStop:finished:context:)];

  /* Анимация заканчивается в нижнем правом углу. */
    [self.xcodeImageView setFrame:CGRectMake(200.0f,
                                             350.0f,
                                             100.0f,
                                             100.0f)];

    [UIView commitAnimations];

  }
```

6. Далее выполните реализацию метода делегата `imageViewDidStop:finished:context:` для контроллера вида, чтобы он вызывался UIKit по завершении анимации. Это не обязательно, так что для примера я просто зарегистрирую несколько сообщений, демонстрирующих, что метод действительно был вызван. В следующих примерах будет показано, как можно использовать метод для запуска какой-то иной активности в момент окончания анимации:

```
- (void)imageViewDidStop:(NSString *)paramAnimationID
                finished:(NSNumber *)paramFinished
                 context:(void *)paramContext{

  NSLog(@"Animation finished.");

  NSLog(@"Animation ID = %@", paramAnimationID);

  UIImageView *contextImageView = (__bridge UIImageView *)paramContext;
  NSLog(@"Image View = %@", contextImageView);

}
```

Теперь, запустив приложение, вы заметите, что, как только отобразится вид, изображение, показанное на рис. 17.33, начнет перемещаться в нижний правый угол (рис. 17.34). На это уйдет 5 секунд.

Кроме того, обратив внимание на консоль и дождавшись окончания анимации, вы увидите примерно следующий текст:

```
Animation finished.
Animation ID = xcodeImageViewAnimation
Image View = <UIImageView: 0x8eaee20;
frame = (220468; 100100); opaque = NO;
userInteractionEnabled = NO;
layer = <CALayer: 0x8eaef10>>
```

А теперь рассмотрим конкретные концепции и разберемся, как именно мы анимировали этот вид с изображением. Далее перечислены важные методы класса, относящиеся к `UIView`, о которых нужно знать, занимаясь анимацией с UIKit.

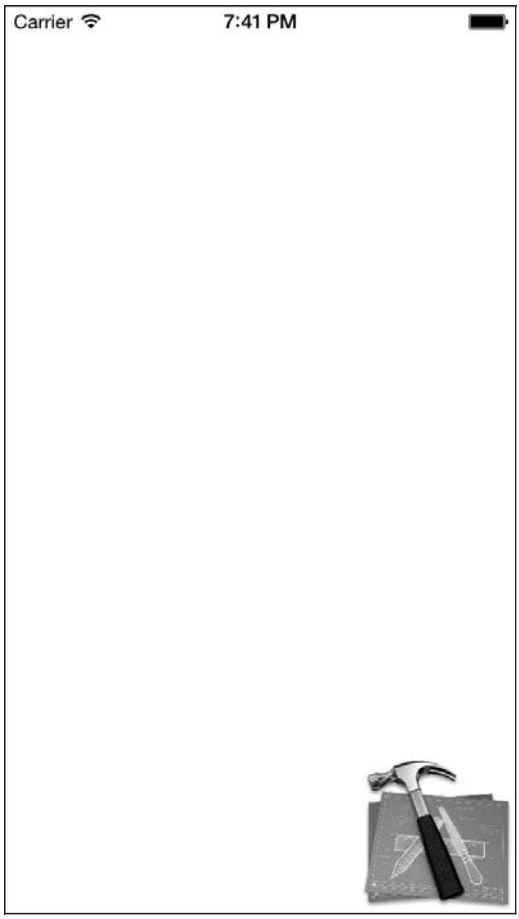


**Рис. 17.34.** Анимируемое изображение переходит в правый нижний угол экрана

- `beginAnimations:context:` — запускает анимационный блок. Любое анимируемое изменение свойств, которое вы применяете к видам после вызова этого метода класса, будет вступать в силу после выполнения анимации.
- `setAnimationDuration:` — этот метод задает длительность анимации в секундах.
- `setAnimationDelegate:` — задает объект, который будет получать сообщения делегатов, касающиеся различных событий, которые могли произойти до, во время или после анимации. Если мы задаем объект делегата, это не означает, что анимационные делегаты немедленно запускаются. Кроме того, вы должны использовать различные методы-установщики, относящиеся к классу, применяя их к объекту вида. Так вы сообщаете UIKit, какие селекторы в вашем объекте-делегате какие делегатные сообщения должны получать.
- `setAnimationDidStopSelector:` — задает в объекте-делегате метод, который должен быть вызван после завершения анимации. Этот метод должен принимать три параметра в следующем порядке:

1) идентификатор анимации типа NSString: здесь будет содержаться идентификатор анимации, передаваемый с началом анимации методу класса beginAnimations:context:, относящемуся к классу UIView;
2) индикатор «завершения» типа NSNumber: этот параметр содержит в NSNumber логическое значение. Среда времени исполнения устанавливает его в YES, если анимация была остановлена в коде, не успев полностью завершиться. Если это значение равно NO, то это означает, что анимация была без перерывов воспроизведена до самого конца;
3) контекст типа void *: это контекст, который с началом анимации передается методу класса beginAnimations:context:, относящемуся к классу UIView.

- setAnimationWillStartSelector: — задает селектор, который должен быть вызван в объекте делегата перед самым началом анимации. Селектор, передаваемый этому методу класса, должен иметь два параметра в таком порядке:
  1) идентификатор анимации типа NSString: среда времени исполнения задает для этого параметра значение идентификатора анимации, передаваемого с началом анимации методу класса beginAnimations:context:, относящемуся к классу UIView;
  2) контекст типа void *: это контекст, который с началом анимации был передан методу класса beginAnimations:context:, относящемуся к классу UIView.
- setAnimationDelay: — задает задержку для анимации (в секундах) перед ее началом. Например, если это значение установлено в 3.0f, то анимация будет начинаться через 3 секунды после выполнения этого метода.
- setAnimationRepeatCount: — указывает количество прогонов анимации, которые должны быть выполнены в блоке кода.

Теперь, когда нам известны наиболее полезные методы класса UIView, помогающие анимировать виды, рассмотрим другую анимацию. В этом примере кода я создам два вида с изображениями (в каждом из них будет показано одно и то же изображение), и они появятся на экране в одно и то же время, одно в левом верхнем углу, другое — в правом нижнем (рис. 17.35).

В этом примере изображение из верхнего левого угла будет называться image 1, а из правого нижнего — image 2.

Как уже упоминалось, в этом коде мы собираемся создать два изображения, в верхнем левом и правом нижнем углах. Далее image 1 станет двигаться по направлению к image 2 и будет так перемещаться на протяжении 3 секунд, а потом медленно исчезнет. Когда image 1 начнет движение, станет двигаться и image 2 — оно пойдет в верхний левый угол экрана, где изначально находилось изображение image 1. Опять же мы хотим, чтобы анимация изображения image 2 завершилась за 3 секунды и оно медленно исчезло. Когда вы запустите этот код на устройстве или симуляторе iOS, такая анимация будет выглядеть *очень классно*. Теперь расскажу, как все это запрограммировать.

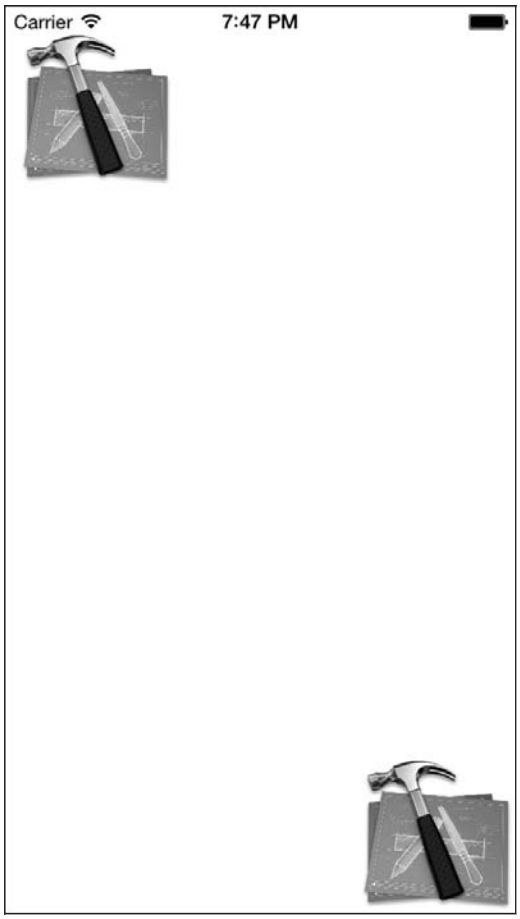


**Рис. 17.35.** Исходное положение, с которого начинается анимация

1. В верхней части .m-файла нашего контроллера вида определим два вида с изображениями:

```
@interface ViewController ()
@property (nonatomic, strong) UIImageView *xcodeImageView1;
@property (nonatomic, strong) UIImageView *xcodeImageView2;
@end

@implementation ViewController
```

2. В методе экземпляра viewDidLoad, относящемся к контроллеру вашего вида, инициализируем оба этих вида с изображениями и помещаем их в основной вид:

```
- (CGRect) bottomRightRect{
CGRect endRect;
endRect.origin.x = self.view.bounds.size.width - 100;
endRect.origin.y = self.view.bounds.size.height - 100;
endRect.size = CGSizeMake(100.0f, 100.0f);
```

```
return endRect;
}
- (void) viewDidLoad{
  [super viewDidLoad];

  UIImage *xcodeImage = [UIImage imageNamed:@"Xcode.png"];

  self.xcodeImageView1 = [[UIImageView alloc]
                          initWithImage:xcodeImage];

  self.xcodeImageView2 = [[UIImageView alloc]
                          initWithImage:xcodeImage];

  /* Просто задаем размеры так, чтобы изображения уменьшились. */
  [xcodeImageView1 setFrame:CGRectMake(0.0f,
                                       0.0f,
                                       100.0f,
                                       100.0f)];

[self.xcodeImageView2 setFrame:[self bottomRightRect]];

  self.view.backgroundColor = [UIColor whiteColor];
  [self.view addSubview:self.xcodeImageView1];
  [self.view addSubview:self.xcodeImageView2];

}
```

3. Реализуем для нашего контроллера вида метод экземпляра, который называется `startTopLeftImageViewAnimation`. Как понятно из названия[1], данный метод будет выполнять анимацию для изображения `image 1`, перемещая его из верхнего левого угла экрана в нижний правый, а изображение тем временем будет медленно исчезать. Такое исчезновение достигается установкой альфа-значения в `0`:

```
- (void) startTopLeftImageViewAnimation{

  /* Начинаем с верхнего левого угла. */
  [self.xcodeImageView1 setFrame:CGRectMake(0.0f,
                                            0.0f,
                                            100.0f,
                                            100.0f)];
  [self.xcodeImageView1 setAlpha:1.0f];

  [UIView beginAnimations:@"xcodeImageView1Animation"
                  context:(__bridge void *)self.xcodeImageView1];

  /* Трехсекундная анимация */
  [UIView setAnimationDuration:3.0f];

  /* Получаем анимационные делегаты. */
  [UIView setAnimationDelegate:self];
```

[1] StartTopLeft (англ.) — «начинаем с верхнего левого угла», ImageView (англ.) — «вид с изображением», Animation (англ.) — «анимация». — *Примеч. пер.*

```
    [UIView setAnimationDidStopSelector:
    @selector(imageViewDidStop:finished:context:)];

    /* Заканчиваем в нижнем правом углу. */
    [self.xcodeImageView1 setFrame:CGRectMake(220.0f,
                                              350.0f,
                                              100.0f,
                                              100.0f)];

    [self.xcodeImageView1 setAlpha:0.0f];

    [UIView commitAnimations];

}
```

4. Когда анимация какого-либо из этих видов остановится, мы удалим данный вид из иерархии родительских видов, так как больше в нем не нуждаемся. Как было показано в методе `startTopLeftImageViewAnimation`, мы передали селектор делегата методу класса `setAnimationDidStopSelector:`, относящемуся к классу `UIView`. Этот селектор будет вызываться после окончания анимации `image 1` (как было показано ранее) и `image 2` (как мы вскоре увидим). Вот реализация этого селектора делегата:

```
- (void)imageViewDidStop:(NSString *)paramAnimationID
                finished:(NSNumber *)paramFinished
                 context:(void *)paramContext{

    UIImageView *contextImageView = (__bridge UIImageView *)paramContext;
    [contextImageView removeFromSuperview];

}
```

5. Кроме того, нам понадобится метод для анимирования `image 2`. Между написанием анимационных методов для `image 2` и `image 1` есть небольшая разница. Я хочу начать анимацию `image 2`, *немного не дожидаясь* завершения анимации `image 1`. Следовательно, если анимация `image 1` завершается за 3 секунды, то я начну анимировать `image 2` со второй секунды анимации `image 1`. Таким образом, анимация `image 2` начнется еще до того, как изображение `image 1` дойдет до нижнего правого угла экрана и исчезнет. Чтобы достичь такого результата, я установлю начало анимации для обоих изображений на одно и то же время, но перед началом анимации `image 2` поставлю двухсекундную задержку. Итак, если обе анимации начнутся в час дня, то для изображения `image 1` начальным моментом анимации будет 13:00:00, а конечным — 13:00:03. Соответствующие значения `image 2` будут равны 13:00:02 и 13:00:05. Вот как будет происходить анимация `image 2`:

```
- (void) startBottomRightViewAnimationAfterDelay:(CGFloat)paramDelay{

    /* Начинаем с нижнего правого угла. */
    [self.xcodeImageView2 setFrame:[self bottomRightRect]];

    [self.xcodeImageView2 setAlpha:1.0f];
```

```
    [UIView beginAnimations:@"xcodeImageView2Animation"
                    context:(__bridge void *)self.xcodeImageView2];

    /* Трехсекундная анимация */
    [UIView setAnimationDuration:3.0f];
    [UIView setAnimationDelay:paramDelay];

    /* Получаем анимационные делегаты. */
    [UIView setAnimationDelegate:self];

    [UIView setAnimationDidStopSelector:
    @selector(imageViewDidStop:finished:context:)];

    /* Заканчиваем в верхнем левом углу. */
    [self.xcodeImageView2 setFrame:CGRectMake(0.0f,
                                              0.0f,
                                              100.0f,
                                              100.0f)];

    [self.xcodeImageView2 setAlpha:0.0f];

    [UIView commitAnimations];

}
```

6. И последнее, но немаловажное замечание. Как только вид отобразится, мы должны запустить методы `startTopLeftImageViewAnimation` и `startBottomRightViewAnimationAfterDelay:`:

```
- (void) viewDidAppear:(BOOL)paramAnimated{

    [super viewDidAppear:paramAnimated];
    [self startTopLeftImageViewAnimation];
    [self startBottomRightViewAnimationAfterDelay:2.0f];

}
```

# 17.15. Анимирование и масштабирование видов

## Постановка задачи

Требуется возможность анимировать виды и масштабировать их в сторону увеличения или уменьшения.

## Решение

Создайте для вида аффинное преобразование и используйте анимационные методы `UIView` для сопровождения масштабирования анимацией.

## Обсуждение

Перед дальнейшей работой настоятельно рекомендую перечитать раздел 17.14.

Чтобы масштабировать вид, анимируя его при этом, можно либо применить к виду преобразование масштабирования в анимационном блоке (см. раздел 17.12), либо просто увеличить высоту и/или ширину вида.

Рассмотрим, как изменять масштаб вида, применяя к нему преобразование масштабирования:

```
- (void) viewDidAppear:(BOOL)paramAnimated{
  [super viewDidAppear:paramAnimated];

  /* Помещаем вид с изображением в центре основного вида данного
     контроллера вида. */
  self.xcodeImageView.center = self.view.center;

  /* Убеждаемся, что к этому виду с изображением не применяется никакого
     преобразования сдвига. */
  self.xcodeImageView.transform = CGAffineTransformIdentity;

  /* Начинаем анимацию. */
  [UIView beginAnimations:nil
                  context:NULL];

  /* Анимация продлится 5 секунд. */
  [UIView setAnimationDuration:5.0f];

  /* Вдвое увеличиваем вид с изображением в ширину и в длину. */
  self.xcodeImageView.transform = CGAffineTransformMakeScale(2.0f,
                                                             2.0f);

  /* Выполняем анимацию. */
  [UIView commitAnimations];

}
```

В этом коде используется аффинное преобразование масштабирования, в результате которого вид с изображением становится в два раза больше по сравнению с исходными размерами. Самое большое достоинство такой операции заключается в том, что в ходе масштабирования начало координат (центр) при увеличении или уменьшении совпадает с началом координат (центром) самого вида. Предположим, что центр вашего вида расположен на экране в точке с координатами (100; 100), а вы хотите масштабировать вид, вдвое увеличив его ширину и высоту. В результате центр вида так и останется в точке (100; 100), в то время как сам вид увеличится в два раза. Если бы мы увеличивали вид, сначала специально добавив ему ширины, а потом высоты, то вид, который получился бы в итоге, находился бы немного не в той точке экрана, где был исходный вид. Это объясняется тем, что, изменяя высоту и ширину рамок вида, вы одновременно изменяете значения $x$ и $y$ контура вида, хотите вы того

или нет. Поэтому вид с изображением не будет масштабироваться относительно своего центра. Исправление такой проблемы выходит за рамки этой книги, но вы можете самостоятельно разобраться с этой задачей — может быть, вам удастся найти решение. Дам *одну подсказку*: можно параллельно запустить две анимации. Одна из них будет изменять длину и ширину вида, а другая — перемещать центр вида.

## См. также

Разделы 17.12 и 17.14.

# 17.16. Анимирование и вращение видов

## Постановка задачи

Требуется анимировать виды на экране при вращении.

## Решение

Создайте аффинное преобразование вращения, для анимирования вращения пользуйтесь методами класса `UIView`.

Перед дальнейшей работой настоятельно рекомендую перечитать раздел 17.14.

Чтобы вращать вид, анимируя его при этом, нужно применить к нему преобразование вращения в то время, как в коде выполняется анимационный блок (см. раздел 17.12). Рассмотрим пример кода, который прояснит это. Допустим, у нас есть рисунок `Xcode.png` (см. рис. 17.9) и мы хотим отобразить его в центре экрана. После того как картинка появится на экране, мы повернем ее на 90° за 5 секунд, а потом повернем обратно, поставив в исходное положение. Итак, когда вид с изображением появится на экране, повернем этот вид на 90° по часовой стрелке:

```
- (void) viewDidAppear:(BOOL)paramAnimated{
  [super viewDidAppear:paramAnimated];

  self.xcodeImageView.center = self.view.center;

  /* Начинаем анимацию. */
  [UIView beginAnimations:@"clockwiseAnimation"
                  context:NULL];

  /* Анимация будет длиться 5 секунд. */
  [UIView setAnimationDuration:5.0f];

  [UIView setAnimationDelegate:self];

  [UIView setAnimationDidStopSelector:
```

```
  @selector(clockwiseRotationStopped:finished:context:)];

  /* Поворачиваем вид с изображением на 90°. */
  self.xcodeImageView.transform =
  CGAffineTransformMakeRotation((90.0f * M_PI) / 180.0f);

  /* Выполняем анимацию. */
  [UIView commitAnimations];

}
```

Мы решили, что селектор `clockwiseRotationStopped:finished:context:` должен вызываться в тот момент, когда заканчивается анимация вращения по часовой стрелке. В этом методе мы будем вращать вид с изображением против часовой стрелки, обратно в положение, соответствующее 0° (то есть исходное). На это тоже уйдет 5 секунд.

```
- (void)clockwiseRotationStopped:(NSString *)paramAnimationID
                        finished:(NSNumber *)paramFinished
                         context:(void *)paramContext{
  [UIView beginAnimations:@"counterclockwiseAnimation"
                  context:NULL];
  /* 5 секунд */
  [UIView setAnimationDuration:5.0f];

  /* Возврат в исходное положение */
  self.xcodeImageView.transform = CGAffineTransformIdentity;

  [UIView commitAnimations];
}
```

Как было показано в разделах 17.14 и 17.15, а также в этом разделе, существует много способов анимировать виды (прямые или непрямые подклассы `UIView`). При выполнении анимации можно изменять немало свойств. Будьте креативны и экспериментируйте с другими свойствами `UIView`, о которых раньше, возможно, не знали. Не помешает также еще раз пересмотреть документацию по `UIView` в органайзере Xcode.

## См. также

Разделы 17.13–17.15.

# 17.17. Получение изображения со скриншотом вида

## Постановка задачи

Требуется сохранить содержимое объекта-вида, находящегося в вашем приложении, в виде изображения. Возможно, также потребуется сохранить это изображение

на диске и выполнить с ним другое действие — например, позволить пользователю поделиться этой картинкой в любимой социальной сети (см. раздел 11.11).

## Решение

Выполните следующие шаги.

1. Воспользуйтесь функцией `UIGraphicsBeginImageContextWithOptions` для создания нового контекста изображения. Этот контекст сразу станет текущим (актуальным), и именно в нем будет происходить все последующее рисование.
2. Вызовите метод `drawViewHierarchyInRect:` вашего класса `UIView`. В качестве параметра передайте этому методу границы вида, который вы хотите отрисовать в текущем контексте.
3. Вызовите метод `UIGraphicsGetImageFromCurrentImageContext`, возвращаемое значение которого — это представление текущего контекста в качестве изображения. Это изображение будет относиться к типу `UIImage`.
4. Преобразуйте ваш экземпляр изображения в данные, воспользовавшись функцией `UIImagePNGRepresentation`. Эта функция даст вам объект типа `NSData`.
5. Наконец, вызовите в вашем объекте данных метод экземпляра `writeToUrl:atomically:`, чтобы записать изображение на определенный адрес на диске — если хотите. Имея экземпляр `UIImage`, можете выполнить с этим изображением и любую другую операцию.

## Обсуждение

Иногда разработчику требуется программно делать скриншот содержимого, которое отображается на экране устройства. В частности, это может понадобиться, если вы пишете приложение для рисования и хотите предоставить пользователю возможность сохранить сделанный рисунок в файле. Возможно, этот файл затем будет сохранен в облаке iCloud, откуда его можно будет впоследствии загрузить.

Перед тем как сохранить изображение или поделиться им таким образом, мы должны нарисовать его в *контексте изображения*. Контекст изображения остается для нас невидимым, так как мы даже не имеем его описателя. Тем не менее все рисовальные методы, которые вы вызываете, будут оказывать влияние на текущий контекст изображения. Контекст изображения можно сравнить с невидимым холстом для рисования. Чтобы получить представление вашего изображения, воспользуйтесь функцией `UIGraphicsGetImageFromCurrentImageContext`.

Когда вы начнете решать такую задачу с помощью нового SDK, вам всего лишь потребуется вызвать в виде метод `drawViewHierarchyInRect:` — и содержимое этого вида будет отрисовано в текущем контексте.

Итак, применим изученный материал на практике. В следующем фрагменте кода мы собираемся разместить в нашем виде ряд компонентов (при этом используются раскадровки, описанные в главе 6). Не важно, что именно вы поместите в раскадровке. Мы хотим снять содержимое нашего вида, сохранить эту информацию как изображение, а затем поместить это изображение в каталог Documents (Документы) на диске:

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

  /* Делаем скриншот */
  UIGraphicsBeginImageContextWithOptions(self.view.bounds.size, YES, 0.0f);
  if ([self.view drawViewHierarchyInRect:self.view.bounds]){
    NSLog(@"Successfully draw the screenshot.");
  } else {
    NSLog(@"Failed to draw the screenshot.");
  }
  UIImage *screenshot = UIGraphicsGetImageFromCurrentImageContext();
  UIGraphicsEndImageContext();

  /* Сохраняем его на диске */
  NSFileManager *fileManager = [[NSFileManager alloc] init];
  NSURL *documentsFolder = [fileManager URLForDirectory:NSDocumentDirectory
                            inDomain:NSUserDomainMask
                            appropriateForURL:nil
                            create:YES
                            error:nil];
  NSURL *screenshotUrl = [documentsFolder
  URLByAppendingPathComponent:@"screenshot.png"];

  NSData *screenshotData = UIImagePNGRepresentation(screenshot);

  if ([screenshotData writeToURL:screenshotUrl atomically:YES]){
    NSLog(@"Successfully saved screenshot to %@", screenshotUrl);
  } else {
    NSLog(@"Failed to save screenshot.");
  }

}
```

В начале этого кода мы создаем новый контекст изображения и получаем его представление в виде изображения с помощью `UIGraphicsGetImageFromCurrentImageContext`. Имея это представление, мы воспользуемся `NSFileManager`, чтобы найти путь к каталогу Documents (Документы) нашего приложения, который находится на диске (см. раздел 12.1). Наконец, мы получаем представление скриншота в виде данных (с помощью функции `UIImagePNGRepresentation`) и после этого можем сохранить данное представление на диске. Мы должны получить представление изображения в формате PNG или JPEG, воспользовавшись для этого функцией `UIImageJPEGRepresentation`. Так мы получим данные, соответствующие изображению в этом формате (PNG/JPEG). Имея данные, мы можем сохранить их на диске или выполнить с ними другие операции.

## См. также

Раздел 11.11, глава 6.

# 18 Фреймворк Core Motion

## 18.0. Введение

Устройства с операционной системой iOS, в частности iPhone и iPad, обычно оборудованы акселерометром. На некоторых устройствах, например новых iPhone и iPad, есть также гироскоп. Прежде чем пытаться использовать в ваших приложениях для iOS акселерометр или гироскоп, нужно проверить доступность (наличие) этих сенсоров на том устройстве, на котором работает ваша программа. В разделах 18.1 и 18.2 описаны приемы, которые можно использовать для обнаружения акселерометра или гироскопа. Устройства iOS, оснащенные гироскопом, могут регистрировать движение вдоль шести осей.

Рассмотрим ситуацию, которая позволяет оценить, насколько полезен гироскоп. Например, акселерометр не может обнаружить вращение устройства вокруг его вертикальной оси, если вы крепко держите устройство в руках, сидите в компьютерном кресле и крутитесь на нем по часовой стрелке или против часовой стрелки. Относительно пола в вашей комнате или относительно планеты Земля устройство вращается вокруг вертикальной оси, но оно при этом не вращается вокруг собственной оси *Y*, проходящей по вертикали через центр устройства, то есть акселерометр не обнаружит никакого движения.

Гироскоп, имеющийся в некоторых устройствах с iOS, может регистрировать такие движения. И мы можем писать более гладкие и безошибочные программы обнаружения движения. Обычно такие возможности полезны в играх, так как при их программировании разработчику зачастую требуется узнать не только о том, как устройство движется по осям *X*, *Y* и *Z* — эти данные можно получить от акселерометра, — но и о том, движется ли устройство по этим осям относительно Земли. А вот для этого уже нужен гироскоп.

Программист может пользоваться фреймворком Core Motion для доступа к информации, поступающей как от акселерометра, так и от гироскопа (при их наличии). Фреймворк Core Motion применяется во всех разделах этой главы. При работе с новым компилятором LLVM, чтобы связать новое приложение с системным фреймворком, вам всего лишь потребуется импортировать этот фреймворк в верхней части

заголовочных файлов и файлов реализации, а компилятор сам выполнит все необходимые операции для импорта фреймворка в приложение.

Эмулятор iOS не может имитировать работу акселерометра и гироскопа. Правда, в эмуляторе iOS можно имитировать *встряхивание*, выбрав команду Hardware ▸ Shake Gesture (Оборудование ▸ Жест встряхивания) (рис. 18.1).

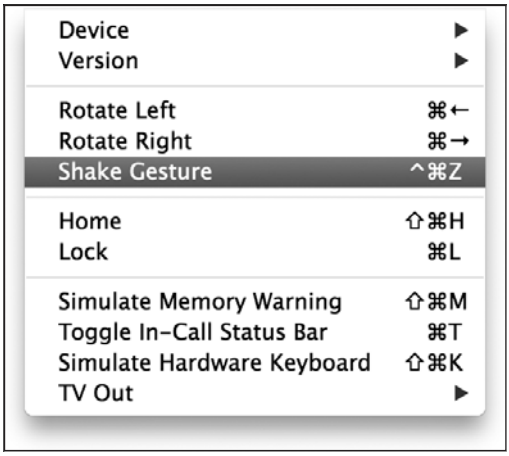


**Рис. 18.1.** Параметр Shake Gesture (Жест встряхивания) в эмуляторе iOS

# 18.1. Обнаружение доступности акселерометра

## Постановка задачи

В вашей программе требуется определить, имеется ли в устройстве акселерометр.

## Решение

Для обнаружения акселерометра пользуйтесь методом `isAccelerometerAvailable` класса `CMMotionManager`. Метод `isAccelerometerActive` также позволяет узнать, посылает ли акселерометр в данный момент уведомления вашей программе.

Сначала убедимся, что импортировали необходимые заголовочные файлы:

```
#import "AppDelegate.h"
#import <CoreMotion/CoreMotion.h>

@implementation AppDelegate
```

Далее проверим, что присутствие акселерометра указано в файле реализации делегата нашего приложения:

```
- (BOOL)            application:(UIApplication *)application
  didFinishLaunchingWithOptions:(NSDictionary *)launchOptions{

  CMMotionManager *motionManager = [[CMMotionManager alloc] init];

  if ([motionManager isAccelerometerAvailable]){
    NSLog(@"Accelerometer is available.");
  } else{
    NSLog(@"Accelerometer is not available.");
  }

  if ([motionManager isAccelerometerActive]){
    NSLog(@"Accelerometer is active.");
  } else {
    NSLog(@"Accelerometer is not active.");
  }

  self.window = [[UIWindow alloc] initWithFrame:
                 [[UIScreen mainScreen] bounds]];

  self.window.backgroundColor = [UIColor whiteColor];
  [self.window makeKeyAndVisible];
  return YES;
}
```

Итак, в устройстве с iOS, где работает ваша программа, может присутствовать акселерометр. Но это еще не означает, что акселерометр посылает уведомления вашему приложению. Если акселерометр или гироскоп *посылает* такие уведомления, мы говорим, что он *активен* (а в таком случае нам потребуется определить объект делегата, о чем поговорим чуть позже).

Если запустить этот код в эмуляторе iOS, то в окне консоли появятся примерно такие сообщения:

```
Accelerometer is not available.  // акселерометр недоступен
Accelerometer is not active.     // акселерометр неактивен
```

При запуске такого же кода на новом iPhone получим такие значения:

```
Accelerometer is available.      // акселерометр доступен
Accelerometer is not active.     // акселерометр неактивен
```

## Обсуждение

В устройстве с операционной системой iOS может быть встроенный акселерометр. Поскольку мы не можем с уверенностью сказать, в каких устройствах с iOS имеется такое оборудование, а в каких — нет, перед использованием акселерометра целесообразно проверить, доступен ли он.

Чтобы проверить наличие этого оборудования, нужно инстанцировать объект типа `CMMotionManager` и получить доступ к его методу `isAccelerometerAvailable`. Это метод логического типа, он возвращает значение `YES`, если акселерометр доступен, и `NO`, если он отсутствует.

Кроме того, можно узнать о том, посылает ли акселерометр обновления вашей программе в настоящий момент (соответственно, активен ли он), воспользовавшись методом `isAccelerometerActive` класса `CMMotionManager`. О том, как получать данные от акселерометра, мы поговорим в разделе 18.3.

## См. также

Раздел 18.3.

# 18.2. Обнаружение доступности гироскопа

## Постановка задачи

В вашей программе требуется определить, имеется ли в устройстве гироскоп.

## Решение

Пользуйтесь методом `isGyroAvailable`, относящимся к классу `CMMotionManager`, чтобы проверить наличие гироскопа. Кроме того, метод `isGyroActive` позволяет узнать, посылает ли в данный момент гироскоп обновления вашему приложению, то есть активен ли он:

```
#import "AppDelegate.h"
#import <CoreMotion/CoreMotion.h>

@implementation AppDelegate

- (BOOL)            application:(UIApplication *)application
  didFinishLaunchingWithOptions:(NSDictionary *)launchOptions{

  CMMotionManager *motionManager = [[CMMotionManager alloc] init];

  if ([motionManager isGyroAvailable]){
    NSLog(@"Gryro is available.");
  } else {
    NSLog(@"Gyro is not available.");
  }

  if ([motionManager isGyroActive]){
    NSLog(@"Gryo is active.");
  } else {
    NSLog(@"Gryo is not active.");
  }

  self.window = [[UIWindow alloc] initWithFrame:
                 [[UIScreen mainScreen] bounds]];

  self.window.backgroundColor = [UIColor whiteColor];
```

```
  [self.window makeKeyAndVisible];
  return YES;
}
```

Эмулятор iOS не позволяет имитировать работу гироскопа. Запустив этот код в эмуляторе, вы увидите в окне консоли примерно такой текст:

```
Gyro is not available.// гироскоп недоступен
Gyro is not active.      // гироскоп неактивен
```

Если запустить этот код на устройстве с iOS, оборудованном гироскопом, например на новом iPhone, то результаты будут иными:

```
Gyro is available. // гироскоп доступен
Gyro is not active.   // гироскоп неактивен
```

## Обсуждение

Если вы планируете выпустить приложение, в котором используется гироскоп, то нужно гарантировать, что ваша программа сможет работать и на других устройствах с iOS, где нет такого оборудования. Например, если вы используете гироскоп как элемент игры, то нужно убедиться в том, что игра будет работать и на других устройствах, хотя гироскопа они и не имеют. Ведь не во всех устройствах с iOS он установлен. Именно наличие гироскопа в устройстве мы и будем проверять в данном разделе.

Чтобы решить эту задачу, потребуется инстанцировать объект типа `CMMotionManager`. После этого мы должны будем получить доступ к логическому методу `isGyroAvailable` и посмотреть, доступен ли гироскоп на том устройстве, где выполняется ваш код. Кроме того, можно воспользоваться методом `isGyroActive` экземпляра `CMMotionManager`, чтобы узнать, посылает ли гироскоп в настоящее время обновления вашему приложению. Подробнее об этом поговорим в разделе 18.5.

## См. также

Раздел 18.5.

# 18.3. Получение данных акселерометра

## Постановка задачи

Требуется указать операционной системе iOS, чтобы она посылала вашей программе данные от акселерометра.

## Решение

Пользуйтесь методом экземпляра `startAccelerometerUpdatesToQueue:withHandler:`, относящимся к классу `CMMotionManager`. Вот заголовочный файл контроллера вида,

в котором класс CMMotionManager применяется для получения обновлений от акселерометра:

```
#import "ViewController.h"
#import <CoreMotion/CoreMotion.h>

@interface ViewController ()
@property (nonatomic, strong) CMMotionManager *motionManager;
@end

@implementation ViewController
```

Мы реализуем контроллер вида и воспользуемся методом startAccelerometer-UpdatesToQueue:withHandler: класса CMMotionManager:

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

  self.motionManager = [[CMMotionManager alloc] init];

  if ([self.motionManager isAccelerometerAvailable]){
    NSOperationQueue *queue = [[NSOperationQueue alloc] init];
    [self.motionManager
     startAccelerometerUpdatesToQueue:queue
     withHandler:^(CMAccelerometerData *accelerometerData, NSError *error) {
       NSLog(@"X = %.04f, Y = %.04f, Z = %.04f",
             accelerometerData.acceleration.x,
             accelerometerData.acceleration.y,
             accelerometerData.acceleration.z);
     }];
  } else {
    NSLog(@"Accelerometer is not available.");
  }
}
```

## Обсуждение

Акселерометр фиксирует данные по трем измерениям (то есть по осям декартовых координат), которые iOS сообщает вашей программе как значения *x*, *y* и *z*. Эти значения инкапсулируются в структуре CMAcceleration:

```
typedef struct {
    double x;
    double y;
    double z;
} CMAcceleration;
```

Предположим, что вы держите устройство с iOS прямо перед собой, экран обращен к вам и находится в книжной ориентации. В таком случае:

- ось *X* расположена слева направо и проходит по центру экрана устройства. При этом значения изменяются слева направо в диапазоне от –1 до +1;

- ось *Y* расположена снизу вверх и проходит по центру экрана устройства. При этом значения изменяются снизу вверх в диапазоне от –1 до +1;
- ось *Z* проходит через заднюю плоскость устройства, потом через все устройство и через экран — по направлению к вам. При этом значения изменяются от задней до передней плоскости устройства в диапазоне от –1 до +1.

Значения, принимаемые от акселерометра, лучше всего разобрать на примерах. Предположим, что вы держите устройство с iOS вертикально экраном к себе. Его нижняя сторона обращена вниз, верхняя — вверх. Если вы будете держать устройство совершенно ровно, не наклоняя его ни в одну из сторон, в этот момент по осям *X*, *Y* и *Z* вы зафиксируете следующие значения: $x = 0{,}0$; $y = -1{,}0$; $z = 0{,}0$. А теперь примем это положение за исходное и попробуем выполнить следующие манипуляции.

1. Повернем устройство на 90° по часовой стрелке. В этот момент вы зафиксируете значения $x = +1{,}0$; $y = 0{,}0$; $z = 0{,}0$.
2. Повернем устройство еще на 90° по часовой стрелке. В данный момент верхняя сторона устройства должна указывать вниз. При этом вы зафиксируете значения $x = 0{,}0$; $y = +1{,}0$; $z = 0{,}0$.
3. Повернем устройство еще на 90° по часовой стрелке. В данный момент верхняя сторона устройства должна указывать влево. При этом вы зафиксируете значения $x = -1{,}0$; $y = 0{,}0$; $z = 0{,}0$.
4. Наконец, если еще раз повернем устройство на 90° по часовой стрелке, так, чтобы верхняя сторона устройства опять была направлена вверх, а нижняя — вниз, то мы вернемся к исходным значениям $x = 0{,}0$; $y = -1{,}0$; $z = 0{,}0$.

Таким образом, можно сделать вывод, что при вращении устройства вокруг оси *Z* меняются значения *x* и *y*, сообщаемые акселерометром, а значение *z* остается неизменным.

Проведем другой эксперимент. Снова расположим устройство горизонтально, так, чтобы его задняя поверхность была обращена вниз, передняя — вверх. Как вы уже знаете, в таком случае акселерометр зафиксирует значения $x = 0{,}0$; $y = -1{,}0$; $z = 0{,}0$. А теперь попробуйте выполнить следующие манипуляции.

1. Наклоните устройство назад на 90° по оси *X* так, чтобы его верхняя сторона указывал назад, то есть держите его так, как будто оно лежит на столе экраном вверх. В этот момент вы зафиксируете значения $x = 0{,}0$; $y = 0{,}0$; $z = -1{,}0$.
2. Теперь поверните устройство еще на 90° назад, так, чтобы задняя поверхность была обращена к вам, верхняя сторона была направлена вниз, а нижняя — вверх. В этот момент акселерометр зафиксирует значения $x = 0{,}0$; $y = 1{,}0$; $z = 0{,}0$.
3. Поверните устройство еще на 90° назад. Теперь его экран должен смотреть вниз, задняя поверхность — вверх, а верхняя сторона должна быть направлена к вам. В этот момент акселерометр должен показывать значения $x = 0{,}0$; $y = 0{,}0$; $z = 1{,}0$.
4. И наконец, если еще раз повернуть устройство в том же направлении, чтобы экран был направлен к вам, верхняя сторона устройства — вверх и т. д., то акселерометр покажет исходные значения, с которых мы начали второй опыт.

Итак, можно сделать вывод, что при вращении устройства вокруг оси *X* изменяются значения по осям *Y* и *Z*, но не по оси *X*. Можете попробовать и третий тип вращения — по оси *Y* (она идет сверху вниз) — и посмотреть, как изменяются значения по осям *X* и *Z*.

Получать обновления от акселерометра можно двумя способами.

- Пользоваться методом экземпляра `startAccelerometerUpdatesToQueue:withHandler:`, относящимся к классу `CMMotionManager`. Этот метод будет доставлять обновления, поступающие от акселерометра, в рабочую очередь (здесь мы имеем дело с очередью типа `NSOperationQueue`). Для работы с ним нужно иметь базовое представление о блоках, которые активно используются при работе с Grand Central Dispatch (GCD). Подробнее о блоках рассказано в главе 7.
- Пользоваться методом экземпляра `startAccelerometerUpdates`, относящимся к классу `CMMotionManager`. Как только вы вызовете этот метод, акселерометр (при его наличии) начнет обновлять свои данные в объекте менеджера движений (Motion Manager). Нужно создать отдельный поток для непрерывного считывания значений свойства `accelerometerData` (типа `CMAccelerometerData`) класса `CMMotionManager`.

В этом разделе мы использовали первый подход (с применением блоковых объектов). Прежде чем продолжать работу с этим разделом, рекомендую внимательно изучить главу 7. Блок, который мы предоставляем методу экземпляра `startAccelerometerUpdatesToQueue:withHandler:`, относящемуся к классу `CMMotionManager`, должен быть объектом типа `CMAccelerometerHandler`:

```
typedef void (^CMAccelerometerHandler)
             (CMAccelerometerData *accelerometerData, NSError *error);
```

Иными словами, блок должен принимать два параметра. Первый параметр должен относиться к типу `CMAccelerometerData`, второй — к типу `NSError`. Так мы и сделали в приведенном примере кода.

## См. также

Раздел 18.1.

# 18.4. Обнаружение встряхивания устройства с iOS

## Постановка задачи

Необходимо узнавать, когда пользователь встряхивает устройство с iOS.

## Решение

Пользуйтесь методом `motionEnded:withEvent:`. Он может относиться к любому объекту вашего приложения, если этот объект принадлежит к типу `UIResponder`. Так, это могут быть контроллеры видов и даже объект основного окна.

## Обсуждение

Метод `motionEnded:withEvent:` окна вашего приложения вызывается всякий раз, когда операционная система iOS фиксирует движение. Простейшая реализация этого метода такова:

```
- (void) motionEnded:(UIEventSubtype)motion
          withEvent:(UIEvent *)event{

  /* Обрабатываем движение. */

}
```

Как видите, параметр `motion` относится к типу `UIEventSubtype`. Тип `UIEventSubtype` имеет, в частности, значение `UIEventSubtypeMotionShake`, которое нас и интересует. Зарегистрировав такое событие, мы можем быть уверены в том, что пользователь встряхнул устройство.

Далее переходим к реализации контроллера вида и обрабатываем метод `motionEnded:withEvent:`:

```
- (void) motionEnded:(UIEventSubtype)motion withEvent:(UIEvent *)event{
  if (motion == UIEventSubtypeMotionShake){
    UIAlertView *alert =
    [[UIAlertView alloc] initWithTitle:@"Shake"
                               message:@"The device is shaken"
                              delegate:nil
                     cancelButtonTitle:@"OK" otherButtonTitles:nil];
    [alert show];
}
}
```

Если теперь встряхнуть устройство или имитировать такое движение в эмуляторе iOS (см. введение к этой главе), в окне консоли мы увидим текст Detected a shake (Обнаружено встряхивание).

# 18.5. Получение данных гироскопа

## Постановка задачи

Требуется получать информацию о движении устройства от гироскопа, установленного в устройстве с iOS.

## Решение

Выполните следующие шаги.

1. Выясните, имеется ли в данном устройстве гироскоп. О том, как это делается, рассказано в разделе 18.2.

2. Если гироскоп в устройстве есть, проверьте, не посылает ли он уже уведомления. О том, как это делается, рассказано в разделе 18.2.
3. Воспользуйтесь методом экземпляра `setGyroUpdateInterval:`, относящимся к классу `CMMotionManager`, чтобы указать, сколько обновлений вы хотите получать в секунду. Например, если вы желаете получать 20 обновлений в секунду, задайте здесь значение `1.0/20.0`.
4. Активизируйте метод экземпляра `startGyroUpdatesToQueue:withHandler:`, относящийся к классу `CMMotionManager`. Объект очереди может просто представлять собой главную операционную очередь (как мы увидим позже), а блок обработчика должен соответствовать формату `CMGyroHandler`.

   Эти шаги реализуются в следующем коде:

```
- (BOOL)            application:(UIApplication *)application
  didFinishLaunchingWithOptions:(NSDictionary *)launchOptions{
  CMMotionManager *manager = [[CMMotionManager alloc] init];

  if ([manager isGyroAvailable]){
    if ([manager isGyroActive] == NO){
      [manager setGyroUpdateInterval:1.0f / 40.0f];
      NSOperationQueue *queue = [[NSOperationQueue alloc] init];
      [manager
       startGyroUpdatesToQueue:queue
       withHandler:^(CMGyroData *gyroData, NSError *error) {
         NSLog(@"Gyro Rotation x = %.04f", gyroData.rotationRate.x);
         NSLog(@"Gyro Rotation y = %.04f", gyroData.rotationRate.y);
         NSLog(@"Gyro Rotation z = %.04f", gyroData.rotationRate.z);
       }];
    } else {
      NSLog(@"Gyro is already active.");
    }
  } else {
    NSLog(@"Gyro isn't available.");
  }
  self.window = [[UIWindow alloc] initWithFrame:
                 [[UIScreen mainScreen] bounds]];
  self.window.backgroundColor = [UIColor whiteColor];
  [self.window makeKeyAndVisible];
  return YES;
}
```

## Обсуждение

Класс `CMMotionManager` позволяет программисту получить от операционной системы iOS обновления данных гироскопа. Сначала нужно убедиться в том, что гироскоп имеется в том устройстве с iOS, где работает ваше приложение (подробнее об этом рассказано в разделе 18.2). Убедившись в этом, можно вызвать метод экземпляра `setGyroUpdateInterval:`, относящийся к классу `CMMotionManager`, чтобы задать коли-

чество обновлений, которые вы хотите получать от гироскопа в секунду. Например, если вам требуется *N* обновлений в секунду, задайте здесь значение `1.0/N`.

Установив интервал обновлений, можно вызвать метод экземпляра `startGyroUpdatesToQueue:withHandler:`, относящийся к классу `CMMotionManager`, — так задается блок для обработки обновлений. О блоках подробнее рассказано в главе 7. Блоковый объект должен относиться к типу `CMGyroHandler`, принимающему два параметра.

- `gyroData` — данные, поступающие от гироскопа, заключены в объекте типа `CMGyroData`. Можно использовать свойство `rotationRate` класса `CMGyroData` (это структура) для получения доступа к значениям *x*, *y* и *z*. Эти значения представляют три эйлерова угла, называемых соответственно крен, тангаж и рыскание. Подробнее эти углы рассматриваются в работах по аэродинамике.
- `error` — ошибка типа `NSError`, которая может возникнуть, когда гироскоп посылает нам обновления.

Если вы не хотите работать с блоковыми объектами, нужно вызвать метод экземпляра `startGyroUpdates`, относящийся к классу `CMMotionManager`, — вместо метода экземпляра `startGyroUpdatesToQueue:withHandler:` того же класса, — а также создать специальный собственный поток. В этом потоке мы будем считывать обновления, поступающие от гироскопа и передаваемые свойству `gyroData` экземпляра используемого нами класса `CMMotionManager`.

## См. также

Раздел 18.2.

# 19 Фреймворк Pass Kit

## 19.0. Введение

Пожалуй, всем приходилось иметь дело с проездными билетами, дисконтными картами и скидочными талонами. Например, вы ходите в кофейню, в которой постоянным клиентам раздают специальные дисконтные карты. Если вы уже успели заказать определенное количество чашек кофе, то вам предложат бесплатную чашечку за счет заведения. Дисконтные карты используются и в магазинах. Например, вы можете купить в супермаркете продукты на сумму *X*, за что вам выдадут дисконтную карту или скидочный купон, которые можно будет использовать, когда вы в следующий раз придете в этот магазин.

На рис. 19.1 показано, как на устройстве с iOS выглядит обычный (бесплатный) железнодорожный билет, отображенный в Passbook.

Приложения iOS для взаимодействия с такими бесплатными билетами могут использовать также фреймворк Passbook. Вернемся к примеру с кофейней. Если мы пишем мобильное приложение для этой кофейни, то можем предусмотреть в нем и такую возможность: посетитель добавляет к дисконтной карточке некоторую сумму, что позволяет ему не только выпить кофе, но и воспользоваться Wi-Fi-соединением. Итак, когда пользователь открывает такое приложение, он замечает в базе данных Passbook клубный талон, полученный именно в этой кофейне. Пользователь может добавить некоторую сумму к этому талону прямо на устройстве, а затем сообщить барриста, что внес плату за пользование сетью через приложение, где имеется виртуальный клубный талон.

Pass Kit — это цифровое решение от Apple для осуществления именно таких транзакций. Итак, разберемся с терминологией, а затем изучим эту тему подробнее.

- *Pass Kit*. Фреймворк Pass Kit от Apple позволяет разработчикам доставлять виртуальные скидочные талоны, снабженные цифровой подписью, на совместимые устройства с операционной системой iOS 6 и выше.
- *Passbook*. Клиентское приложение на устройствах с iOS 7. Позволяет хранить и обрабатывать талоны, созданные разработчиками, а также управлять этими талонами.


**Рис. 19.1.** Железнодорожный билет, представленный в виде талона в приложении Passbook на устройстве с iOS

Итак, мы (разработчики) будем пользоваться Pass Kit для создания талонов с цифровой подписью и доставки их нашим пользователям. Пользователям потребуется взаимодействовать с талонами, которые мы для них создаем, для этого они будут использовать приложение Passbook, установленное на устройстве. Итак, мы сможем выдавать пользователям в цифровой форме различные талоны, проездные билеты, скидочные купоны, дисконтные и клубные карты и т. д. Не придется носить целую стопку карточек в кошельке. Весь этот контент можно хранить в одном месте — в приложении Passbook для iOS.

Прежде чем опробовать эту новую технологию, рассмотрим общую картину: как именно спроектировано такое хранилище и как оно помогает достигать наших целей. Я разделил эту концепцию на несколько более мелких элементов. Рассмотрим, как доставлять пользователям талоны, снабженные цифровой подписью.

1. Разработчик создает сертификат и соответствующий ему закрытый ключ. Это делается на портале инициализации Apple (Apple Provisioning Portal).
2. Затем разработчик создает ряд файлов. Они будут представлять собой талон, который мы позже доставим пользователю.
3. Разработчик подписывает созданный талон с применением того сертификата, который был создан на первом этапе.
4. Разработчик тем или иным способом доставляет талон пользователю.
5. Пользователь увидит талон и сможет добавить его на свое устройство.
6. Как только талон окажется на пользовательском устройстве, Passbook сохранит его для последующего использования до тех пор, пока пользователь не решит удалить талон.

Конечно, после прочтения этого списка у вас могла еще не сложиться целостная картина. На рис. 19.2 все показано наглядно.

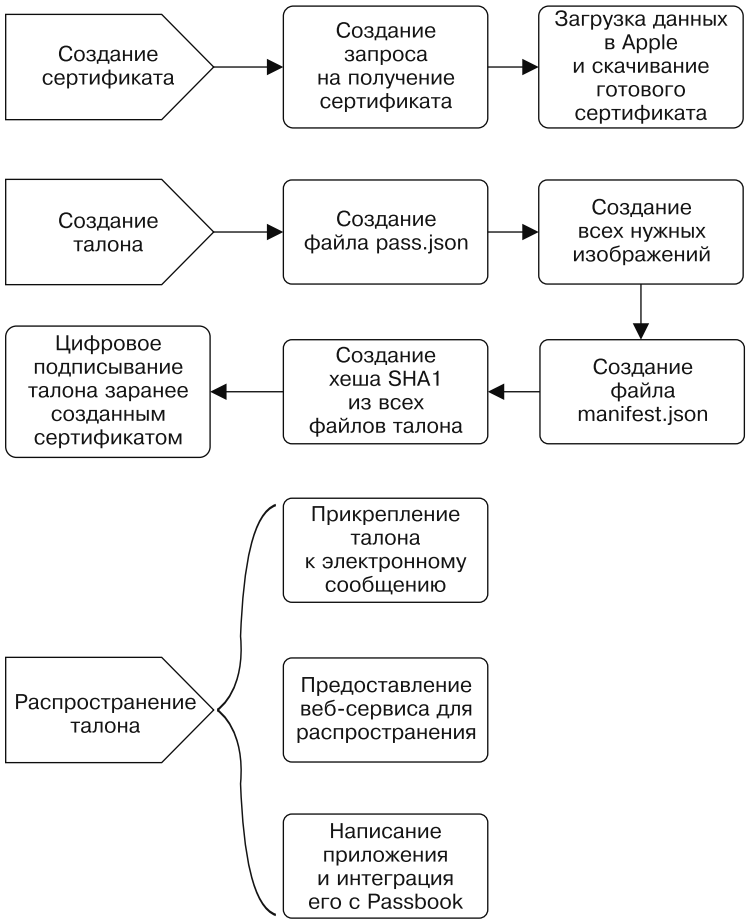


**Рис. 19.2.** Создание и распространение среди пользователей талонов с цифровой подписью в операционных системах iOS 6 и выше

Все детали процесса будут подробно описаны в разделах этой главы. Некоторые технологические аспекты Pass Kit, связанные с тем, как обеспечивать актуальность ваших талонов и отсылать уведомления с сервера, требуют определенных знаний о серверном программировании. Ради упрощения материала мы не будем в этой главе останавливаться на данных аспектах, а поговорим преимущественно о создании талонов. Научившись создавать талоны, вы сможете распространять их разными способами. Два из таких способов объяснены в этой главе. Те детали, которые связаны не с системой iOS непосредственно, а с серверной разработкой, в этой главе не рассматриваются.

# 19.1. Создание сертификатов Pass Kit

## Постановка задачи

Требуется распространять между пользователей талоны с цифровой подписью. Для этого первым делом нужно создать цифровые сертификаты для подписывания ваших талонов.

## Решение

Создавайте цифровые сертификаты на портале инициализации iOS.

## Обсуждение

Как было объяснено во введении к этой главе, для распространения талонов между пользователями эти талоны требуется снабжать цифровыми подписями. А перед этим нужно получить в Apple сертификат, который позволил бы уникально связать все талоны с вашей учетной записью разработчика. Таким образом Apple отличает «настоящие» талоны от «ненастоящих».

При создании сертификата выполните следующие шаги.

1. В браузере откройте центр разработки для iOS (iOS Dev Center). Не буду ставить здесь ссылку, так как со временем она может измениться. Просто введите такой запрос в поисковик — и он выдаст вам нужный сайт в мгновение ока.
2. Если вы еще не вошли на сайт под вашей учетной записью, сделайте это.
3. Оказавшись в системе, перейдите на страницу Certificates, Identifiers & Profiles (Сертификаты, идентификаторы, профили).
4. Перейдите на страницу Identifiers (Идентификаторы) и далее на страницу Pass Type ID (ID типа талона). Соответствующий раздел находится в левой части экрана.
5. Когда вы окажетесь на нужной странице, она, возможно, еще будет пуста. Найдите и нажмите экранную кнопку +.
6. Теперь в поле Description (Описание) введите текст, который будет описывать ID типа вашего талона.

7. В поле Identifier (Идентификатор) введите идентификатор талона в формате обратного доменного имени. Например, если ID вашего приложения — `com.pixolity.testingpasskit`, то для талонов, интегрированных с данным приложением, подойдет идентификатор `pass.pixolity.ios.cookbook.testingpasses`. Разумеется, следует подбирать для идентификаторов талонов такие названия, которые что-то означают в контексте вашего приложения. На практике принято начинать имя идентификатора с `pass`, а затем можно писать все, что хотите. На рис. 19.3 показано, как нужно заполнять поля на этой странице.

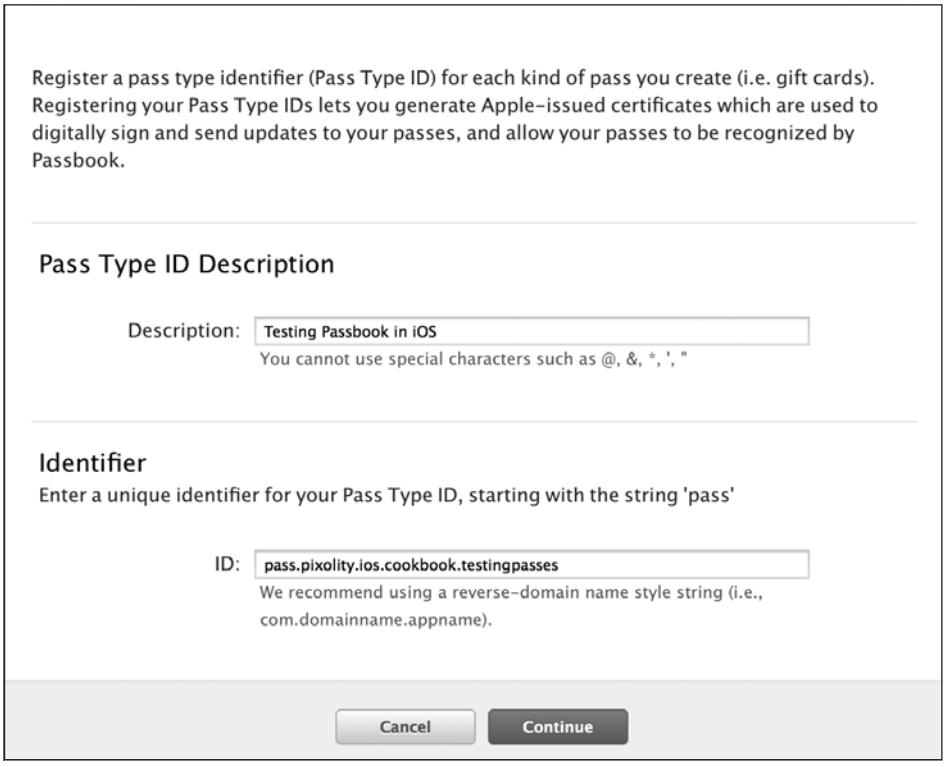


**Рис. 19.3.** Внесение информации о простом идентификаторе типа талона

Когда закончите заполнение этой страницы, нажмите кнопку Continue (Продолжить). Вам для предварительного просмотра будет представлен талон. Если вас все устроит, просто нажмите кнопку Register (Зарегистрировать) (рис. 19.4). Итак, теперь у вас есть идентификатор типа талона. Но он пока не связан ни с одним сертификатом. Чтобы создать сертификат и ассоциировать его с вашим идентификатором типа талона, выполните следующие шаги.

1. В разделе Pass Type ID (Идентификаторы типа талона) на портале инициализации iOS найдите созданный вами идентификатор и нажмите кнопку Settings (Настройки) (рис. 19.5). В столбце Pass Certificate (Сертификаты талонов) в этом списке вы увидите, что ваш идентификатор типа талона обозначен как None

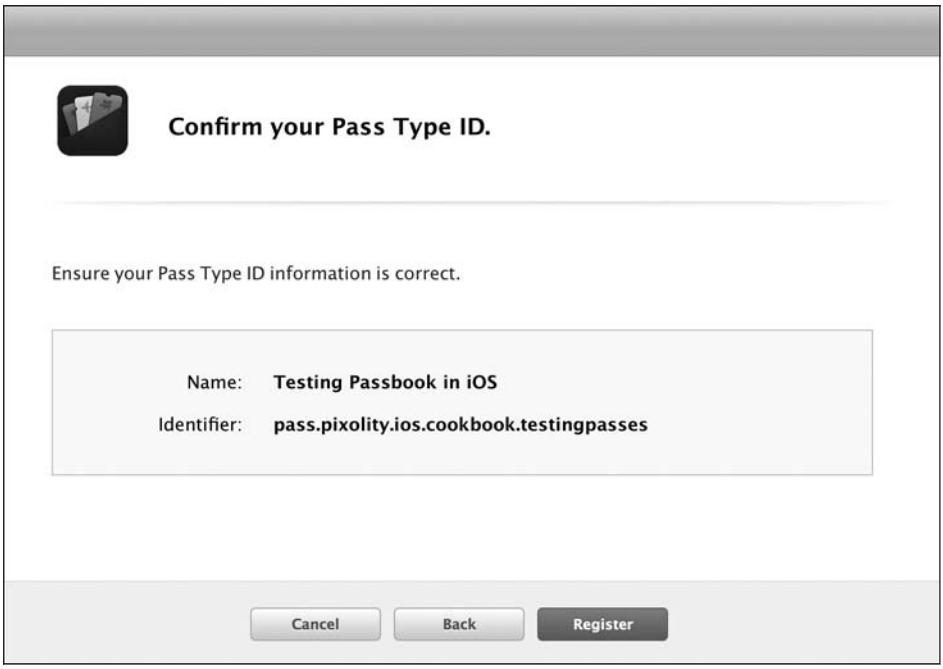


**Рис. 19.4.** Если вся информация указана правильно, подтвердите создание ID типа талона

(Отсутствует). В столбце Action (Действия) нажмите ссылку Configure (Сконфигурировать).

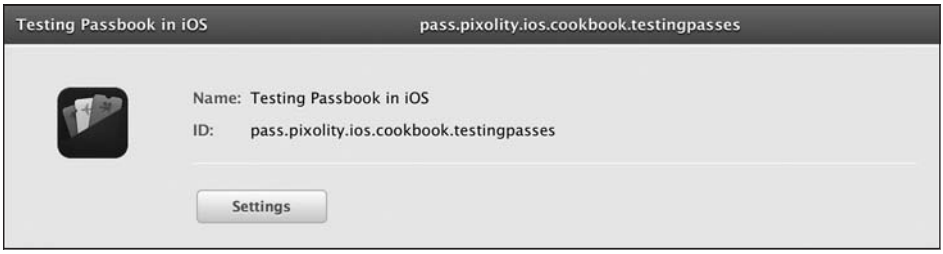


**Рис. 19.5.** Задаем идентификатор типа талона

2. Выбрав на портале идентификатор типа талона, вы сможете создать для него сертификат (рис. 19.6). Просто нажмите кнопку Create Certificate (Создать сертификат).
3. Теперь вам будет предложено создать запрос на подписывание сертификата, воспользовавшись доступом к связке ключей на Mac (рис. 19.7). Следуйте инструкциям, которые видите на экране. Когда закончите создание запроса на подписывание сертификата, нажмите экранную кнопку Continue (Продолжить).


**Рис. 19.6.** Мы готовы создать сертификат для идентификатора типа талона

Запрос на подписывание сертификата можно создать не только в Mac. Для создания такого запроса необходимо убедиться, что на вашей машине установлен Open SSL. Описание генерирования сертификатов на машинах, где не установлен Mac, выходит за рамки этой книги. Однако, если вас интересует этот вопрос, вы легко найдете ответ на него в Интернете — достаточно немного поискать.

Создавая на компьютере запросы на подписывание сертификата (это делается с помощью доступа к связке ключей), вы также создаете закрытый ключ, ассоциированный с таким сертификатом. Apple рекомендует время от времени делать резервные копии базы данных связки ключей, так как на портале инициализации iOS эти ключи не сохраняются. Если вы переходите к работе с новым компьютером, то ваши ключи туда нужно будет перенести вручную. Поэтому они и называются закрытыми. Экспортировать закрытые ключи несложно: щелкните на ключе правой кнопкой мыши и выберите в контекстном меню команду Export (Экспортировать).

4. Далее в браузере вам будет предложено загрузить запрос на подписывание сертификата на сайт Apple, чтобы в ответ получить готовый сертификат. Закрытый ключ был создан на вашем компьютере в тот самый момент, когда вы подготовили запрос на подписывание сертификата. Сертификат, который Apple выдаст по окончании процесса, будет подходить к закрытому ключу. Далее нажмите кнопку Choose File (Выбрать файл) на этом экране, чтобы выбрать запрос на подписывание сертификата, созданный для вас связкой ключей (рис. 19.8). Сделав это, нажмите кнопку Generate (Сгенерировать).


**Рис. 19.7.** Выполняйте инструкции по созданию запроса на подписывание сертификата


**Рис. 19.8.** Загрузка запроса на подписывание сертификата на сайт Apple для получения ссылки на сертификат

5. Когда сертификат сгенерирован, вы увидите примерно такой экран, как на рис. 19.9. Нажмите кнопку Download (Скачать), чтобы скачать сгенерированный сертификат.

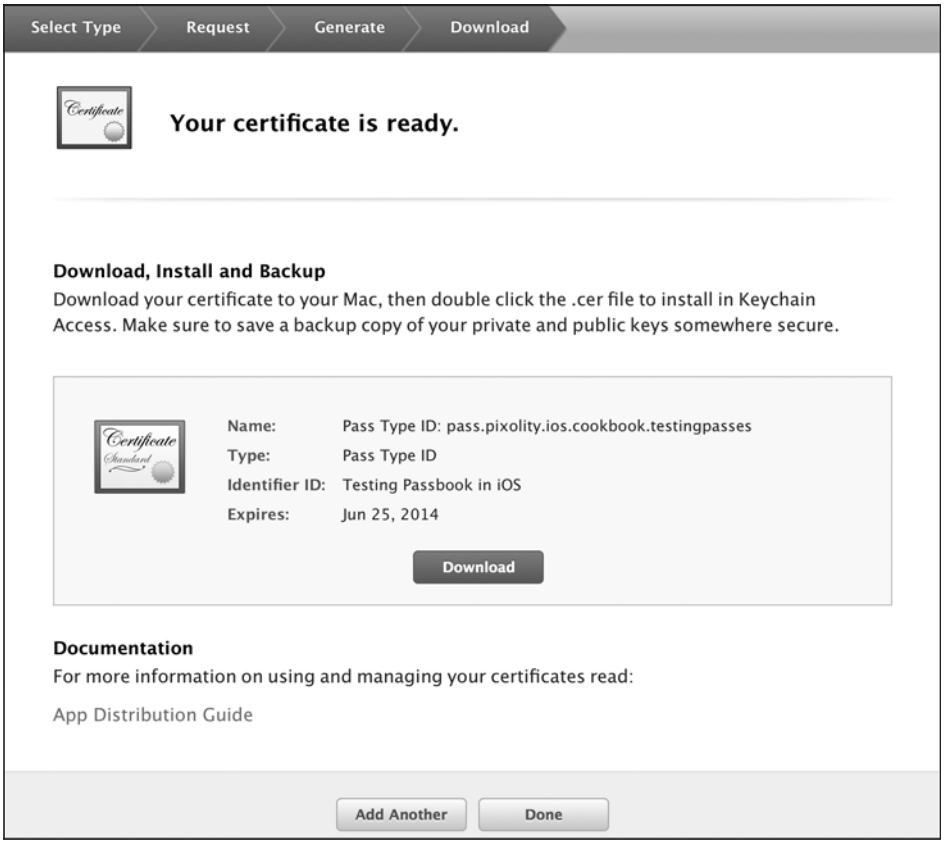


**Рис. 19.9.** Apple сообщает вам о том, что сертификат создан успешно

6. Теперь у вас на диске должен быть готовый сертификат. Найдите этот файл и дважды щелкните на нем кнопкой мыши, чтобы импортировать его в вашу связку ключей. Чтобы убедиться, что все сработало правильно, откройте на компьютере программу для доступа к связке ключей (Keychain Access), там перейдите в раздел Login (Учетная запись) и далее — в подраздел My Certificate (Мой сертификат). Затем в правой части экрана подтвердите, что ваш сертификат на месте и что он ассоциирован с закрытым ключом (рис. 19.10).

Итак, вы создали сертификат и готовы подписывать талоны, а потом рассылать их на устройства с iOS.

## См. также

Раздел 19.0.

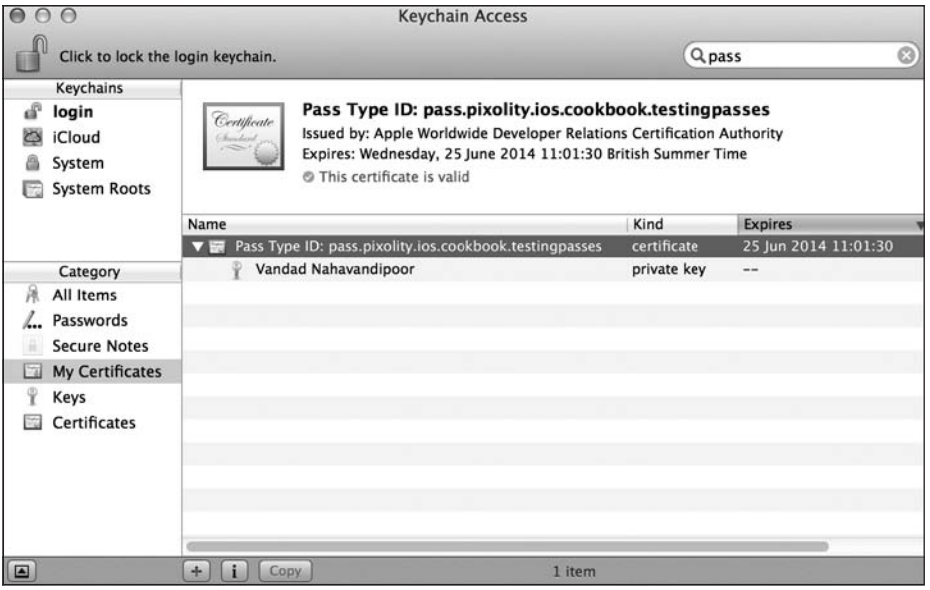


**Рис. 19.10.** Сертификат, сгенерированный Apple, правильно ассоциирован с закрытым ключом

# 19.2. Создание файлов талонов

## Постановка задачи

Требуется создать файл талона, представляющий данные, которые должны сохраняться на пользовательском устройстве с iOS.

## Решение

Создайте файл `pass.json` и заполните его соответствующими ключами и значениями.

## Обсуждение

Для представления талонов во фреймворке Pass Kit компания Apple выбрала формат JSON. Аббревиатура JSON расшифровывается как «объектная нотация JavaScript», широко используется в веб-приложениях и веб-службах. Однако вы, будучи iOS-разработчиком, можете и не иметь достаточных знаний о JSON.

Файлы JSON состоят из обычных ключей и значений, этим они напоминают словари. Ключ может иметь значение, а значения могут быть самыми разными — от простой строки до словаря, который и сам содержит ключи и значения. Вот простой пример JSON, обладающий всеми характерными чертами этой нотации:

```
{
  "key 2 - dictionary" = {
    "key 2.1" = "value 2.1";
```

```
    "key 2.2" = "value 2.2";
  };
  "key 3 - array" = (
  {
    "array item 1, key1" = value;
    "array item 1, key2" = value;
  },
  {
    "array item 2, key1" = value;
    "array item 2, key2" = value;
  }
  );
  key1 = value1;
}
```

Как видите, словари заключены в квадратные скобки, а массивы — в фигурные. Другие элементы представляют собой обычные пары «ключ — значение». Если бы мы попытались представить этот объект JSON в виде экземпляра `NSDictionary`, то у нас получился бы такой код:

```
NSDictionary *json = @{
  @"key1" : @"value1",
  @"key 2 - dictionary" : @{
    @"key 2.1" : @"value 2.1",
    @"key 2.2" : @"value 2.2",
  },
  @"key 3 - array" : @[
  @{
    @"array item 1, key1" : @"value",
    @"array item 1, key2" : @"value"
  },
  @{
    @"array item 2, key1" : @"value",
    @"array item 2, key2" : @"value"
  }
  ]
};
```

Подробнее о нотации JSON вы можете почитать на сайте JSON.org. Перейдем к созданию файлов талонов. Как уже говорилось, такой файл состоит из обычной нотации JSON. Не путайте файлы талонов и сами талоны. Талон — это коллекция файлов, в которую входит и `pass.json`. Вся эта коллекция в целом и будет представлять собой талон с цифровой подписью, который пользователь сможет установить на своем устройстве. Файл талона «объясняет», как талон должен быть представлен на устройстве.

Файл `pass.json` можно создавать с помощью высокоуровневых и низкоуровневых ключей. Высокоуровневыми называются такие ключи, которые сразу же видны в основной иерархии файла `pass.json`. Низкоуровневые ключи являются дочерними для высокоуровневых. Не волнуйтесь, если пока это не совсем понятно. Я тоже сначала запутался в этой иерархии, но читайте дальше — и стройная картина обязательно сложится.

Начнем с создания файла `pass.json` в Xcode. Должен вас предупредить, что Xcode, к сожалению, не лучший инструмент для редактирования JSON. Однако это наша основная интегрированная среда разработки, так что продолжим работать в ней. Чтобы создать файл `pass.json`, выполните следующие шаги.

1. Создайте в Xcode простой проект для iOS, выбрав File ▸ New ▸ Project (Файл ▸ Новый ▸ Проект).
2. В левой части диалогового окна New Project (Новый проект) убедитесь, что находитесь в категории iOS. Затем выберите раздел Other (Другой), а в правой части экрана — вариант Empty (Пустой) (рис. 19.11). Сделав это, нажмите кнопку Next (Далее).


**Рис. 19.11.** Создание пустого проекта в iOS

3. Теперь укажите имя вашего проекта в поле Product Name (Имя продукта). Сделав это, нажмите кнопку Next (Далее). После этого можете сохранить файл на диске. Когда вы успешно выберете путь для сохранения проекта, появится возможность создать файл `pass.json`.
4. В новом пустом проекте в Xcode выберите File ▸ New ▸ File (Файл ▸ Новый ▸ Файл).
5. В диалоговом окне New File (Новый файл), будучи в категории iOS, выберите вариант Other (Другой). Справа выберите вариант Empty (Пустой) (рис. 19.12). Сделав это, нажмите кнопку Next (Далее).
6. После того как вы нажмете кнопку Next (Далее), вам будет предложено сохранить файл на диске. Убедитесь, что сохраняете его как `pass.json`. Справившись с этим, нажмите кнопку Create (Создать), и файл будет добавлен на диск в рамках вашего проекта.


**Рис. 19.12.** Добавление пустого файла в проект

Отлично, вот вы и создали файл `pass.json` на диске. Теперь нужно заполнить этот файл ключами и значениями. Прежде чем мы подробно поговорим о ключах и значениях, детально разберем, какая информация находится в файле талона:

```
{
  "formatVersion" : 1,
  "passTypeIdentifier" : "<# Put your Pass Type ID here #>",
  "serialNumber" : "p69f2J",
  "teamIdentifier" : "<# Put your team ID here #>",
  "description" : "Train Ticket Example",
  "locations" : [
  {
    "longitude" : -0.170867,
    "latitude" : 50.834948
  }
  ],
  "barcode" : {
    "message" : "1234567890",
    "format" : "PKBarcodeFormatPDF417",
    "messageEncoding" : "iso-8859-1"
  },
  "organizationName" : "O'Reilly Railways",
  "logoText" : "O'Reilly Railways",
  "foregroundColor" : "rgb(255, 255, 255)",
  "backgroundColor" : "rgb(100, 100, 100)",
  "boardingPass" : {
```

```
    "transitType" : "PKTransitTypeTrain",
    "primaryFields" : [
    {
      "key" : "departure",
      "label" : "Departs From",
      "value" : "Hove, 07:37",
    },
    {
      "key" : "departurePlatform",
      "label" : "Departs from Platform",
      "value" : "2",
    }
    ],
    "auxiliaryFields" : [
    {
      "key" : "arrival",
      "label" : "Arrives At",
      "value" : "London Bridge, 08:41"
    },
    {
      "key" : "arrivalPlatform",
      "label" : "Arrives at Platform",
      "value" : "13"
    }
    ],
    "backFields" : [
    {
      "key" : "oreillyRailways",
      "label" : "O'Reilly Railways",
      "value" : "For more information, visit www.oreilly.com"
    },
    {
      "key" : "termsAndConditions",
      "label" : "Terms and Conditions",
      "value" : "To be filled later"
    }
    ]
  }
}
```

Я специально оставил ключи teamIdentifier и passTypeIdentifier без значений. Значениями этих ключей должна быть информация, которую вы сами указали на портале инициализации, например идентификатор талона. Значения этих ключей необходимо заполнять совершенно точной собственной информацией.

Класс. Теперь у нас есть готовый файл `pass.json`, который можно включать в талон с цифровой подписью. Не забывайте, талон — это не только файл `pass.json`. В состав талона входят также несколько изображений и файл описания (манифеста), где будут перечислены все файлы, образующие талон.

Приведу некоторые важнейшие ключи, которые могут находиться в файле pass.json:

- formatVersion — этот ключ указывает версию формата талона. Его значение должно быть равно константе 1;
- passTypeIdentifier — это идентификатор талона, созданный вами ранее на портале инициализации iOS, но здесь не указывается ID команды. Например, если мой полный идентификатор типа талона — TEAMID.pass.pixolity.testingpasskit, то в данном случае я укажу значение идентификатора талона как pass.pixolity.testingpasskit;
- teamIdentifier — это идентификатор вашей команды. Чтобы узнать это значение, просто перейдите на главную страницу центра разработки в iOS (iOS Dev Center) и далее — в Центр участников (Member Center). Выберите вашу учетную запись (Your Account), а затем Профиль организации (Organization Profile). Там вы должны найти поле под названием Company/Organization ID (Идентификатор компании/организации). Это идентификатор вашей команды. Просто скопируйте это значение и вставьте его в качестве ключа в ваш файл pass.json;
- description — краткое описание назначения талона. Это описание будет использоваться при оптимизации доступности приложения в iOS;
- organizationName — имя вашей компании;
- serialNumber — уникальный серийный номер талона. Вы можете придумать его по ходу разработки. Он должен быть информативным для вас и вашей организации. Обратите внимание: если два и более талона имеют один и тот же идентификатор типа, то их серийные номера не могут быть одинаковыми;
- barcode — штрихкод для талона. Настоятельно рекомендуется включать в цифровой талон информацию в формате штрихкода. Это словарь, ключи, которые могут в нем находиться, описаны далее:
  - message — сообщение, зашифрованное в штрихкоде;
  - format — формат штрихкода. В качестве значений для этого ключа можно указать PKBarcodeFormatText, PKBarcodeFormatQR, PKBarcodeFormatPDF417 или PKBarcodeFormatAztec. Обсуждение формата штрихкодов выходит за рамки этой книги, поэтому в данном случае не будем вдаваться в детали;
  - messageEncoding — кодировка, применяемая в штрихкоде. В качестве значения этого ключа укажите iso-8859-1;
- logoText — текст, который будет выводиться на вашем талоне рядом с логотипом в приложении Passbook на устройстве;
- foregroundColor — основной цвет вашего талона. Это значение состоит из красного, зеленого и голубого компонентов, каждый из которых может выражаться числом в диапазоне от 0 до 255. Значение включается в функцию rgb(). Например, чистому красному цвету соответствует значение rgb(255, 0, 0), а чистому белому — rgb(255, 255, 255);
- backgroundColor — фоновый цвет вашего талона. Указывается в том же формате, что и foregroundColor.

Когда все нужные значения для этих ключей будут заданы, вы сможете указать тип создаваемого талона. Для этого нужно включить в число высокоуровневых ключей талона либо один из предыдущих ключей, либо один из следующих:

- `eventTicket` — сообщает Passbook, что талон представляет собой билет на мероприятие, например на концерт;
- `coupon` — сообщает Passbook, что талон представляет собой скидочный купон. Например, такой талон может быть выдан в магазине, и пользователь, предъявив его, имеет право на скидку при приобретении тех или иных товаров;
- `storeCard` — говорит Passbook, что талон представляет собой дисконтную или клубную карту;
- `boardingPass` — сообщает Passbook, что талон представляет собой проездной билет на поезд или автобус либо посадочный талон на самолет;
- `generic` — талон, не относящийся ни к одной из вышеупомянутых категорий.

Каждый из приведенных ключей в файле pass.json будет содержать словарь значений (которые, в свою очередь, являются ключами со значениями). Эти ключи будут конкретно определять, для чего применяется талон и какие значения он содержит.

Когда вы внесете талон одного из этих типов в качестве ключа в файл `pass.json`, нужно будет указать словарные ключи и значения для данного талона (мы уже говорили, что все талоны вышеперечисленных типов являются словарями). Каждый словарь такого типа может содержать следующие ключи:

- `transitType` — этот ключ требуется только в словаре типа `boardingPass`. В других случаях его можно просто игнорировать. В этом словаре могут содержаться следующие значения: `PKTransitTypeAir`, `PKTransitTypeBus`, `PKTransitTypeTrain`, `PKTransitTypeBoat` и `PKTransitTypeGeneric`. Талоны с такими значениями соответствуют билетам на самолет, автобус, поезд, водный транспорт. Последнее значение является универсальным;
- `headerFields` — часть информации, расположенная в верхней части талона и доступная для просмотра в Passbook на устройстве. Старайтесь не перегружать этот заголовок информацией, поскольку эти значения всегда будут видны пользователю, даже если все талоны сложены «в стопку» в интерфейсе приложения Passbook;
- `primaryFields` — самая важная информация о вашем талоне, которая будет отображаться на его лицевой стороне. Например, если мы говорим о посадочном талоне на самолет, то здесь вы найдете номер терминала, номер места и название авиакомпании. В другом талоне здесь может присутствовать иной набор значений;
- `secondaryFields` — второстепенная информация, также отображаемая на лицевой стороне талона. Например, в посадочном талоне на самолет к этой категории можно отнести время посадки, дату посадки и тип воздушного судна;
- `auxiliaryFields` — наименее важная информация, отображаемая на лицевой стороне талона. В посадочном талоне на самолет к такой информации можно отнести предположительное время прибытия;
- `backFields` — информация, отображаемая на оборотной стороне талона.

В качестве значений все вышеупомянутые ключи получают словари, а эти словари, в свою очередь, могут содержать следующие ключи:

- `label` — надпись-название поля, которое должно отображаться на талоне (с лицевой или оборотной стороны в зависимости от того, к какому ключу добавлен этот словарь);
- `key` — ключ, которым ваше приложение может воспользоваться для считывания значения этого поля;
- `value` — значение этого поля;
- `textAlignment` — опциональный ключ, который может описывать визуальное выравнивание надписи на талоне. Для этого поля можно указать любое из следующих значений:
  - `PKTextAlignmentRight`;
  - `PKTextAlignmentCenter`;
  - `PKTextAlignmentLeft`;
  - `PKTextAlignmentNatural`;
  - `PKTextAlignmentJustified`.

Да уж, многовато ключей и значений приходится запоминать. Но не волнуйтесь, со временем к этому привыкаешь. Итак, создадим простой файл `pass.json`. Сначала сформулируем требования, а потом приступим к написанию самого файла талона. Как раз применим на практике ту теорию, которую выучили раньше. Далее изложена суть примера.

- Создаваемый талон будет соответствовать железнодорожному билету.
- Поезд выходит из английского города Хоув в 7:37. Состав отбывает от платформы 2.
- Поезд прибывает на вокзал Лондон-Бридж в Лондоне в 8:41 (платформа 13).
- Билет предоставляет право проезда в поездах придуманной нами компании «О'Рейли Рэйлуэйз».

Но прежде, чем приступать к делу, нам потребуется подробно рассмотреть массив `locations` из файла `pass.json`. Этот ключ является массивом, каждый элемент которого имеет по два ключа. Чуть позже мы их рассмотрим. Но самая интересная черта этого ключа заключается в том, что он может содержать геолокационную информацию о создаваемом вами талоне. Когда талон импортируется в приложение Passbook на вашем устройстве с iOS, операционная система выведет для пользователя информацию о вашем талоне. В частности, будет сообщено, что талон действителен в том месте, где сейчас находится пользователь. Предположим следующее: пользователю нужно показывать билет на поезд всякий раз, когда он (как пассажир) подходит к турникету на станции Хоув (станция отправления). Итак, вы можете указать в электронном талоне местоположение станции отправления (по ключу `locations`), чтобы iOS автоматически выводила талон на экран, как только пользователь прибудет на станцию. Вы можете реализовать такую же функцию и для конечной станции, так как, когда вечером пользователь будет уезжать с вокзала Лондон-Бридж в Хоув, станцией отправления станет Лондон. Если вы движетесь

из точки A в точку B, то B — пункт назначения. Когда вы возвращаетесь, B становится точкой отправления, а A — точкой назначения. Итак, вы можете указать в массиве `locations` местоположение точек A и B, а также любых других важных точек на том маршруте, где действует ваш талон. Вот ключи, которые могут входить в состав любого массива с информацией о местоположении:

- `longitude` — долгота географической точки. Это значение типа `double`. Не заключайте его в кавычки;
- `latitude` — широта географической точки. Это значение типа `double`. Не заключайте его в кавычки.

## См. также

Разделы 19.0 и 19.2.

# 19.3. Подготовка пиктограмм и изображений для талонов

## Постановка задачи

Необходимо гарантировать, что оформление вашего талона будет выдержано в стилистике компании. Для этого нужно придать талону характерные черты либо снабдить его узнаваемым изображением.

## Решение

Создайте фон, пиктограммы, логотипы и вставьте их в ваш талон, снабженный цифровой подписью.

## Обсуждение

На талоне могут содержаться различные изображения:

- *фон* (`background.png`, `background@2x.png` и `background-568@2x.png`) — фоновое изображение на талоне. Не на всех талонах есть фоновые изображения;
- *логотип* (`logo.png` и `logo@2x.png`) — логотип, который будет находиться в верхнем левом углу талона. Зависит от типа талона;
- *пиктограмма* (`icon.png` и `icon@2x.png`) — пиктограмма для талона. Не у всех талонов есть пиктограммы. В этой главе мы подробно поговорим о создании пиктограмм для талонов;
- *миниатюра* (`thumbnail.png` и `thumbnail@2x.png`) — миниатюра-ярлык, соответствующая талону. Будет видна, когда талоны сложены «в стопку».

Как понятно из названий, все эти изображения создаются в двух вариантах: для обычного и для сетчатого дисплея (Retina). Apple не требует строгого соблюдения этого правила, но разве мы, разработчики, не ценим наших клиентов? Сетчатые

дисплеи сегодня так популярны, что становятся общепризнанным промышленным стандартом. Поэтому, пожалуйста, не забывайте создавать и сетчатые варианты изображений с высоким разрешением.

Итак, мы выяснили, каковы будут имена файлов изображений. Перейдем к параметрам этих изображений. Я перечисляю только сетчатые изображения (чтобы получить обычные, просто разделите все значения по длине и ширине на 2):

- `background@2x.png` — 640 пикселов в ширину и 960 пикселов в высоту;
- `background-568@2x.png` — 640 пикселов в ширину и 1136 пикселов в высоту, для iPhone 5;
- `logo@2x.png` — 60 пикселов в ширину и 60 пикселов в высоту;
- `icon@2x.png` — 58 пикселов в ширину и 29 пикселов в высоту;
- `thumbnail@2x.png` — 200 пикселов в ширину и 200 пикселов в высоту.

В этом разделе я создал все изображения максимально простым образом. На рис. 19.13 они все показаны на одном холсте.

Рисунок сделан для наглядности и просто показывает, сколько изображений требуется подготовить для одного талона. Не нужно создавать такое изображение, в котором все эти элементы расположены на одном холсте.

Все изображения готовы. Сохраните их в том же каталоге, где уже находится файл `pass.json`. Переходим к следующему этапу работы — создаем файл описания (манифеста).

## См. также

Раздел 19.2.

# 19.4. Подготовка талонов к цифровому подписыванию

## Постановка задачи

Требуется подготовить талоны к цифровому подписыванию. Это необходимый предварительный этап, без которого такое подписывание выполнить невозможно.

## Решение

Создайте файл `manifest.json` в том же каталоге, где находятся файл `pass.json` и изображения для талона. Файл описания будет написан в формате JSON. Его корневой объект — это словарь. Ключами в этом словаре являются имена файлов (имена всех изображений плюс имя файла `pass.json`). Значение каждого ключа представляет собой SHA1-хеш соответствующего файла.

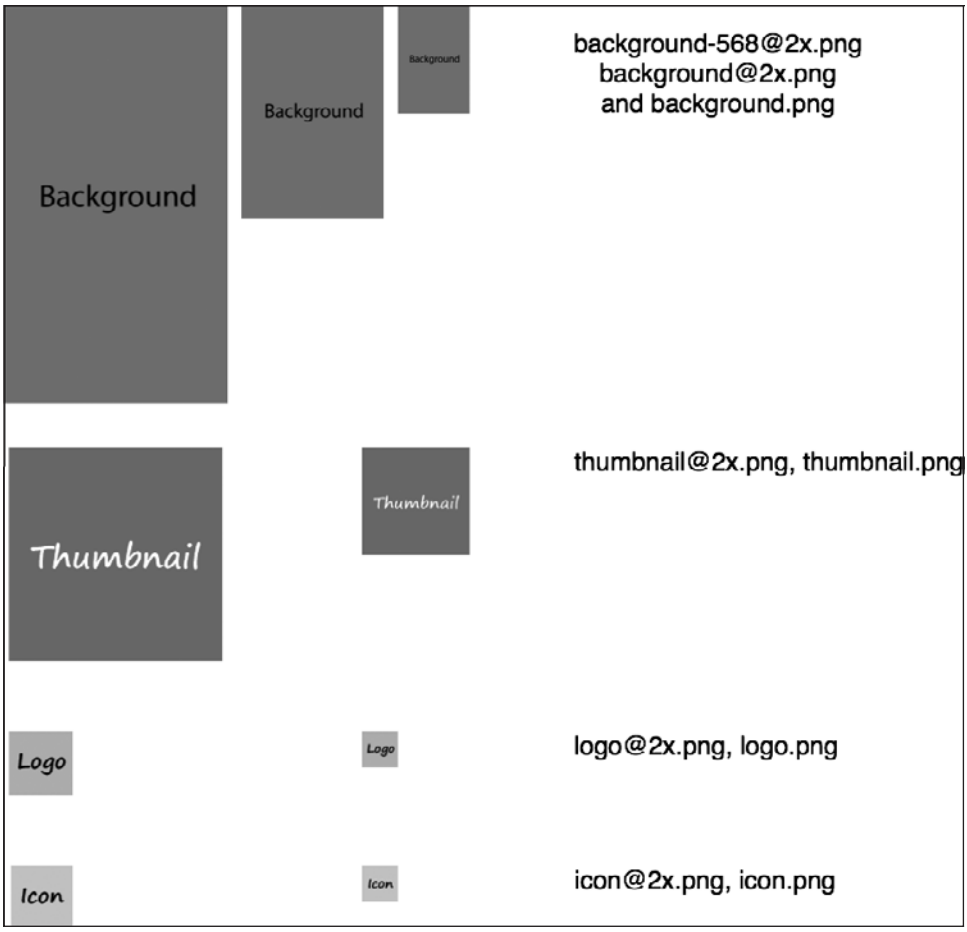


**Рис. 19.13.** Все изображения с талона на одном холсте

## Обсуждение

Просто создайте файл `manifest.json` с ключами для всех изображений, а значения пока оставьте пустыми. Содержимое вашего файла `manifest.json` должно выглядеть примерно так:

```
{
  "background.png" : "",
  "background@2x.png" : "",
  "background-568@2x.png" : "",
  "icon.png" : "",
  "icon@2x.png" : "",
  "logo.png" : "",
  "logo@2x.png" : "",
  "pass.json" : "",
```

```
  "thumbnail.png" : "",
  "thumbnail@2x.png" : ""
}
```

А теперь начинается самое интересное. Нужно рассчитывать SHA1-хеши всех этих файлов. Учтите, что теперь при каждом изменении файлов (например, вы нашли ошибку в файле `pass.json` и исправили ее) потребуется также пересчитывать SHA1-хеш и записывать новое значение этого хеша в файле `manifest.json`. Чтобы рассчитать значение SHA1-хеша для любого файла в операционной системе OS X, просто выполните следующие шаги.

1. Откройте окно терминала и перейдите в каталог, где находится целевой файл (для этого используется команда `cd`).
2. Выполните в окне терминала команду `openssl`. В качестве ее первого аргумента сообщите `sha1`, в качестве второго — имя файла.

Например, в каталоге с моим проектом есть подкаталог `pass`. В этот каталог я поместил мой файл `pass.json`, полупустой файл `manifest.json`, а также все изображения для талона (фоновое изображение, логотип и т. д.). Теперь в окне терминала я рассчитаю значения SHA1-хешей для всех этих файлов и запишу полученные значения в файл описания. Итак, в первой строке следующего листинга идет команда `openssl`, а все остальные строки — это значения хешей, полученные в качестве вывода:

```
openssl sha1 *.png *.json
SHA1(background-568h@2x.png)= e2aaf36f4037b2a4008240dc2d13652aad6a15bb
SHA1(background.png)= b21a92dedb89f8b731adabc299b054907de2347d
SHA1(background@2x.png)= 6abab0f77fd89f1a213940fd5c36792b4cc6b264
SHA1(icon.png)= ed698ab24c5bd7f0e7496b2897ec054bbd426747
SHA1(icon@2x.png)= 90381c84cfea22136c951ddb3b368ade71f49eef
SHA1(logo.png)= c3bd8c5533b6c9f500bbadbdd957b9eac8a6bfe9
SHA1(logo@2x.png)= 1a56a5564dec5e8742ad65dc47aa9bd64c39222f
SHA1(thumbnail.png)= 58883d22196eb73f33ea556a4b7ea735f90a6213
SHA1(thumbnail@2x.png)= 0903df90165ef1a8909a15b4f652132c27368560
SHA1(manifest.json)= 894f795b991681de8b12101afb8c2984bf8d0f65
SHA1(pass.json)= c5acddbab742f488867c34882c55ca14efff0de9
```

Мы рассчитали SHA1-хеши всех файлов, в том числе хеш manifest.json. Однако SHA1-хеш файла manifest.json нам не понадобится, так как он содержит хеши всех остальных файлов, а своего собственного хеша не имеет. Поэтому мы просто игнорируем хеш файла manifest.json.

Итак, теперь требуется заполнить файл `manifest.json` значениями SHA1 всех остальных файлов — эти хеши мы только что рассчитали:

```
{
  "background.png" : "b21a92dedb89f8b731adabc299b054907de2347d",
  "background@2x.png" : "6abab0f77fd89f1a213940fd5c36792b4cc6b264",
  "background-568@2x.png" : "e2aaf36f4037b2a4008240dc2d13652aad6a15bb",
  "icon.png" : "ed698ab24c5bd7f0e7496b2897ec054bbd426747",
  "icon@2x.png" : "90381c84cfea22136c951ddb3b368ade71f49eef",
```

```
  "logo.png" : "c3bd8c5533b6c9f500bbadbdd957b9eac8a6bfe9",
  "logo@2x.png" : "1a56a5564dec5e8742ad65dc47aa9bd64c39222f",
  "pass.json" : "c5acddbab742f488867c34882c55ca14efff0de9",
  "thumbnail.png" : "58883d22196eb73f33ea556a4b7ea735f90a6213",
  "thumbnail@2x.png" : "0903df90165ef1a8909a15b4f652132c27368560"
}
```

Пока все понятно. Переходим к следующему этапу: нам нужно снабдить талон цифровой подписью.

## См. также

Раздел 19.1.

# 19.5. Цифровое подписывание талонов

## Постановка задачи

Вы подготовили каталог `pass` с файлом описания и файлом `pass.json`, а также все изображения. Теперь вы хотите снабдить цифровой подписью каталог с талоном и его содержимое. Это требуется для создания файла талона, готового к распространению.

## Решение

Для подписывания талонов используйте OpenSSL.

## Обсуждение

Каждый талон требуется подписывать с помощью сертификата, созданного в разделе 19.1. Для подписывания талонов мы вновь будем использовать команду `openssl` в окне терминала. Перед тем как читать дальше, убедитесь в том, что создали каталог `pass` и поместили в него файлы `pass.json`, `manifest.json` и все изображения. Этот каталог не обязательно должен называться `pass`. Тем не менее, чтобы было удобнее читать этот раздел и оставшиеся разделы этой главы, лучше не импровизировать и назвать этот каталог с файлами именно `pass`.

Некоторые читатели могли запутаться в том, куда относятся некоторые ключи и для чего нужны те или иные сертификаты. Надеюсь, в этом разделе ситуация прояснится. Когда вы запрашиваете новый сертификат на портале инициализации iOS, в связке ключей на вашем компьютере создается закрытый ключ, а также файл запроса на подпись сертификата (CSR). Сертификат будет сгенерирован Apple. Когда вы скачаете файл сертификата, он будет иметь расширение .cer. Это просто сертификат! Когда вы импортируете его в связку ключей, результирующий файл будет иметь расширение .p12. В этом файле будут содержаться и сертификат, и закрытый ключ к нему.

Прежде чем мы вплотную займемся процессом подписывания, нам потребуется экспортировать созданный сертификат из связки ключей. Не забывайте, что сертификат, полученный вами на портале инициализации iOS, — это не тот самый сертификат, который экспортируется из связки ключей. Поэтому при экспорте сертификата для идентификатора типа талона обязательно выполните следующие шаги.

1. Откройте на Mac программу для доступа к связке ключей (Keychain Access).
2. В левой верхней части окна в области Keychains (Связки ключей) убедитесь, что вы выбрали связку ключей для входа в систему (Login Keychain).
3. В области Category (Категория) слева выберите My Certificates (Мои сертификаты).
4. Найдите ваш сертификат для идентификатора типа талона в правой части экрана и щелкните на нем правой кнопкой мыши.
5. Теперь выберите функцию Export (Экспортировать) (рис. 19.14) и завершите экспорт вашего файла на диск в виде файла `.p12`. Не сохраняйте сертификат в каталоге `pass`. Сертификат должен находиться вне этого каталога.


**Рис. 19.14.** Экспорт сертификата идентификатора типа талона из программы Keychain Access

6. После того как вы попытаетесь экспортировать свой сертификат, система запросит у вас два пароля: пароль, который требуется задать для сертификата, и пароль вашей учетной записи пользователя в OS X — это должен быть пользователь, владеющий доступом к связке ключей. Первый пароль, сопровождающий сертификат, гарантирует, что без этого пароля сертификат нельзя будет импортировать на произвольную машину. Второй пароль удостоверяет, что человек, импортирующий сертификат из связки ключей, действительно имеет на это право. Например, если вы отойдете от компьютера, оставив его незаблокированным, а ваш приятель попытается экспортировать сертификат из вашей

связки ключей, для этого ему придется указать пароль вашей учетной записи. В любом случае рекомендуется гарантировать, что у разных учетных записей в одной системе разные пароли. Например, если вы с братом работаете на одном и том же Mac, то ваш пароль должен быть уникальным именно для вашей учетной записи. Если пароль у вас будет одинаковым, это поставит под сомнение всю систему безопасности вашего компьютера.

Убедитесь, что не сохраняете сертификат в каталоге pass. Нельзя рассылать сертификат вместе с талоном.

Итак, вы экспортировали сертификат. У вас должен был получиться файл, который, возможно, называется `Certificates.p12`. Теперь нужно разделить этот файл на сертификат как таковой и закрытый ключ. Однако если вы подписываете талон с помощью OpenSSL, то вам приходится передавать сертификат и закрытый ключ по отдельности. Чтобы получить из файла `Certificates.p12` закрытый ключ и сертификат (этот файл мы только что экспортировали из программы Keychain Access), выполните следующее.

1. Откройте окно терминала, если еще не сделали этого.
2. Перейдите в каталог, в котором вы сохранили экспортированный сертификат (файл `.p12`).
3. Чтобы получить сертификат, введите в окне терминала следующую команду:

```
openssl pkcs12 -in "Certificates.p12" -clcerts -nokeys \
-out "exported-certificate"
```

После того как вы введете эту команду, система сразу же потребует у вас присвоить экспортированному сертификату пароль. В этом примере я задаю пароль 123, но вы придумайте пароль получше.

4. Чтобы получить закрытый ключ из сертификата, экспортированного из связки ключей, выполните в окне терминала следующую команду:

```
openssl pkcs12 -in "NAME OF YOUR .P12 CERTIFICATE FILE" \
-nocerts -out "NAME OF THE OUTPUT KEY"
```

Я назвал экспортированный закрытый ключ `exported-key`, но вы можете назвать его и иначе, если желаете:

```
openssl pkcs12 -in "Certificates.p12" -nocerts -out "exported-key"
```

От вас вновь потребуется ввести пароли для ключа. Для моего ключа я указал пароль `1234`, чтобы его можно было легко запомнить. Такой же пароль я задал для сертификата. Но если вы работаете в организации, где безопасности уделяется должное внимание, такой пароль вам, разумеется, не подойдет. Выберите пароль, который кажется вам осмысленным. Для максимальной защиты необходимо гарантировать, что все пароли для экспортируемых вами сертификатов/ключей являются уникальными.

Замечательно. Теперь у нас есть файлы экспортированного сертификата и закрытого ключа. Можем перейти к подписыванию талона с их помощью. Чтобы это сделать, выполните следующие шаги.

1. Если вы еще не сделали этого, поместите все файлы, относящиеся к талону (`pass.json`, `manifest.json` и все соответствующие изображения), в каталог `pass`. Вы можете назвать этот каталог как хотите, но в рамках этого раздела рекомендую давать создаваемым каталогам такие же названия, какие даю я. Так нам будет проще ориентироваться, в каком каталоге мы находимся и что делаем в окне терминала.
2. Воспользуйтесь командой `cd`, чтобы перейти из текущего рабочего каталога в каталог `pass`, где находятся все интересующие нас файлы талона.
3. Выполните команду `rm -f .DS_Store`, чтобы убедиться, что в каталоге `pass` отсутствуют ненужные скрытые системные файлы OS X. Вы также должны проверить, перечислены ли все файлы из этого каталога в файле `manifest.json`, где наряду с самими файлами должны находиться и их SHA1-хеши. Если в этот каталог просочатся какие-то другие файлы (как перечисленные в файле описания, так и не указанные там), то талон получится недействительным. Его не удастся прочитать в приложении Passbook ни на устройстве с iOS, ни на симуляторе.
4. Выполните в окне терминала следующую команду, чтобы сгенерировать в каталоге `pass` файл `signature`:

```
openssl smime -binary -sign -signer "PATH TO YOUR EXPORTED CERTIFICATE" \
-inkey "PATH TO YOUR EXPORTED PRIVATE KEY" -in manifest.json \
-out signature -outform DER
```

Эта команда должна выполняться в каталоге pass, где находятся все ваши ресурсы, связанные с талоном. Экспортированный сертификат и закрытый ключ — это тот сертификат и тот ключ, которые были извлечены из сертификата, экспортированного из связки ключей. Не указывайте в этой команде сертификат в том виде, в каком он был экспортирован из связки ключей. Ранее мы уже рассмотрели, как извлекать реальный сертификат и закрытый ключ из файла .p12, экспортированного из связки ключей. Можете перечитать этот фрагмент раздела, чтобы проверить, все ли вы делаете правильно.

На последнем этапе система запросит у вас пароль к закрытому ключу. Помните его? Вы задавали этот пароль, когда извлекали закрытый ключ из сертификата, экспортированного из связки ключей. Теперь эта команда создаст файл `signature` в каталоге `pass`. Почти все готово. Остается заархивировать подготовленный каталог в виде файла с расширением `.pkpass`. Для этого выполните следующие шаги.

1. Откройте окно терминала и с помощью команды `cd` перейдите в каталог `pass`.
2. Чтобы запаковать ваш каталог `pass` в файл `pass.pkpass`, выполните в текущем каталоге следующую команду:

```
zip -r pass.pkpass . -x '.DS_Store'
```

В результате все файлы талона будут упакованы в архив `pass.pkpass`. Вновь необходимо убедиться в том, что в готовом архиве не будет файла `.DS_Store`.

## См. также

Разделы 19.1 и 19.4.

# 19.6. Распространение талонов по электронной почте

## Постановка задачи

Требуется рассылать талоны с цифровой подписью клиентам по электронной почте.

## Решение

Посылайте талоны по электронной почте в виде прикрепленных файлов.

## Обсуждение

Талон, который вы подписали и упаковали в разделе 19.5, готов к распространению. Один из простейших способов распространения талонов — по электронной почте. Выполните следующие шаги, чтобы рассылать ваши талоны с помощью приложения Mail.app из операционной системы OS X.

1. Откройте приложение Mail.app в операционной системе OS X. В меню File (Файл) выберите пункт New Message (Новое сообщение).
2. Введите адрес электронной почты того клиента, которому хотите послать талон.
3. Введите заголовок электронного сообщения.
4. Введите текст электронного сообщения. После этого просто перетащите файл `pass.pkpass`, подготовленный в разделе 19.5, в конец вашего электронного сообщения (рис. 19.15).
5. Отправьте электронное письмо.

В операционной системе OS X Mavericks у пользователей появилась возможность просматривать талоны прямо в приложении Mail.app. Поскольку программа Passbook интегрирована с облаком iCloud, теперь вы можете нажать на сенсорном экране на талон, прикрепленный к электронному письму, и отправить его прямо на ваше устройство (устройства) с iOS. Для этого на мобильном устройстве должна быть установлена программа Passbook, а также активизирована связь с iCloud — это делается в соответствующем разделе настроек (рис. 19.16).

Теперь, работая с сенсорным экраном, пользователь может нажать пальцем на талон, прикрепленный к письму. В результате откроется приложение Passbook, в интерфейсе которого талон отобразится, и пользователь сможет добавить его в Passbook.

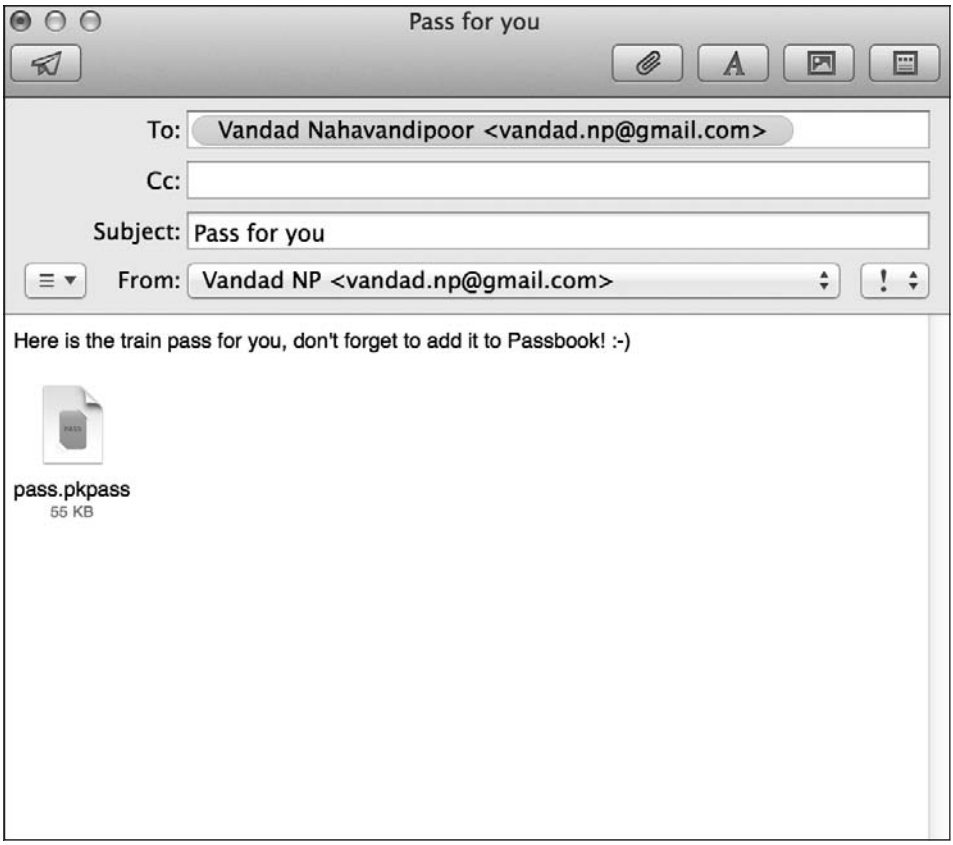


**Рис. 19.15.** Распространение талонов с цифровой подписью с помощью приложения Mail.app операционной системы OS X

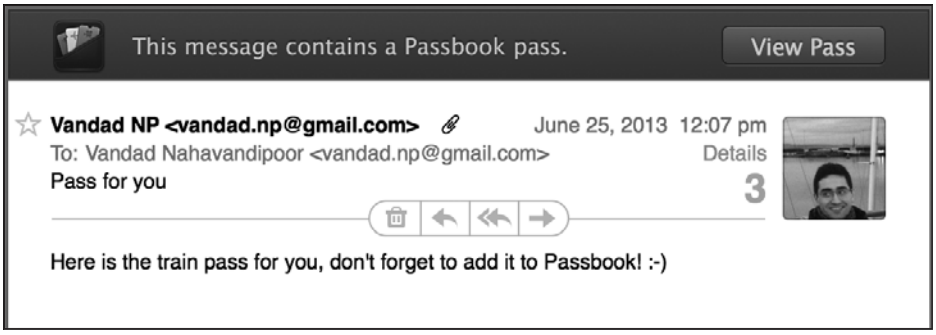


**Рис. 19.16.** В операционной системе OS X Mavericks талоны отображаются прямо в приложении Mail.app

## См. также

Раздел 19.7.

# 19.7. Распространение талонов с помощью веб-сервисов

## Постановка задачи

Требуется, чтобы пользователи могли скачивать прямо с вашего сайта талоны, снабженные цифровой подписью.

## Решение

На своих веб-страницах создавайте гиперссылки на талоны `.pkpass`. Когда пользователь будет просматривать веб-страницы на устройстве, он может просто нажать на сенсорном экране заинтересовавшую его ссылку. Когда он это сделает, браузер Safari обнаружит, что ссылка указывает на файл `.pkpass`, и передаст этот файл в Passbook. В результате талон будет отображаться на сайте, а пользователи смогут добавлять талоны в приложения Passbook.

## Обсуждение

Браузер Safari в операционной системе iOS не может непосредственно обрабатывать загрузку файлов талонов в формате `.pkpass`. Чтобы реализовать такую возможность, необходимо создавать веб-страницы и включать в них гиперссылки, указывающие на файлы `.pkpass`. Простой код на языке HTML, доставляющий пользователю файл `pass.pkpass`, приведен далее:

```
<html>
<header>
  <title>Passbook Site</title>
</header>
<body>
  <a href="http://localhost:8888/pass.pkpass">Download your pass here</a>
</body>
</html>
```

Ссылка указана здесь как localhost, так как в моей операционной системе OS X установлен и работает веб-сервер Apache. Вместо localhost вам понадобится дать такую ссылку, которая будет корректна в вашей среде разработки.

Теперь, когда пользователь откроет ссылку на своем устройстве в браузере Safari, он увидит примерно такую картинку, как на рис. 19.17.

Когда пользователь нажмет на ссылку, перед ним на экране откроется знакомый графический интерфейс Passbook. Пользователь сможет добавить полученный талон в Passbook.

## См. также

Раздел 19.6.

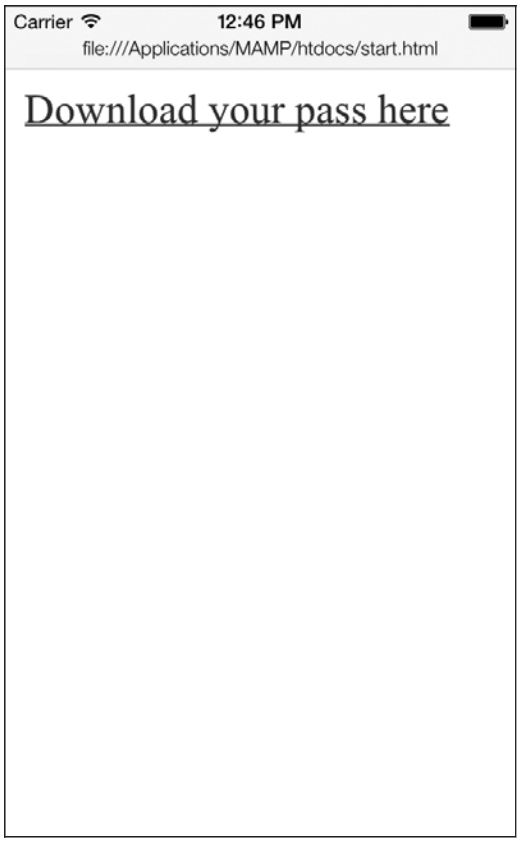


**Рис. 19.17.** Просмотр сайта в браузере Safari на симуляторе iOS

# 19.8. Настройка возможности доступа к талонам в приложениях, работающих на устройстве с операционной системой iOS

## Постановка задачи

Требуется развернуть приложение с поддержкой Passbook на устройстве с операционной системой iOS и гарантировать, что ваше приложение сможет читать талоны с цифровой подписью, имеющиеся на этом устройстве.

## Решение

Создайте соответствующий профиль инициализации для вашего приложения. Он должен быть связан с ID приложения. Это приложение должно обладать доступом на работу с талонами.

## Обсуждение

Необходимо подписывать приложения соответствующим профилем инициализации, который создан на том же портале, что и ID типа талона. Это требуется для того, чтобы вы могли считывать собственные талоны из приложения Passbook, установленного на пользовательском устройстве. Процесс проиллюстрирован на рис. 19.18.

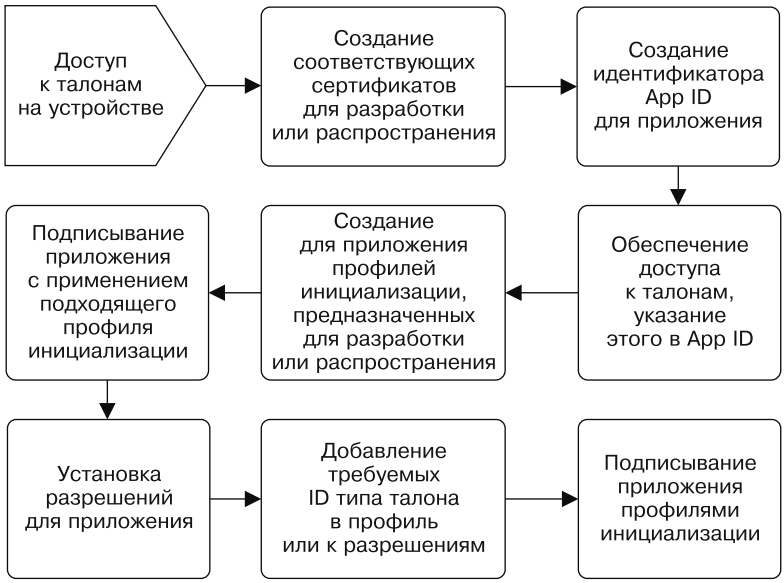


**Рис. 19.18.** Процесс обеспечения доступа к талонам в приложении для iOS на устройстве с iOS

Итак, начнем! Предполагается, что на данном этапе у вас уже есть сертификат на разработку/распространение. Создадим идентификатор App ID для идентификатора типа талона (Pass Type ID), подготовленного нами в разделе 19.1. После этого займемся созданием соответствующего профиля инициализации для App ID. Сделайте вот что.

1. Перейдите в центр разработки для iOS (iOS Dev Center) в вашем браузере и зайдите на этот сайт под своим именем, если еще не сделали этого.
2. Перейдите в раздел Certificates, Identifiers & Profiles (Сертификаты, идентификаторы, профили).
3. Далее перейдите в раздел Identifiers (Идентификаторы), а затем в раздел App ID (Идентификатор приложения) и там нажмите экранную кнопку с символом +.
4. В поле Description (Описание) опишите ID вашего приложения. Эти сведения должны быть информативными для вас, вашей команды и организации.
5. В качестве значения Bundle Seed ID (Префикс идентификатора) укажите Use Team ID (Использовать ID команды).
6. В разделе Bundle Identifier (Идентификатор пакета) (суффикс App ID) укажите идентификатор пакета, записанный в стиле обратного доменного имени. В моем

случае при работе с талоном, имеющим ID `pass.pixolity.testingpasskit`, идентификатор пакета App ID будет записываться как `com.pixolity.testingpasskit`.

7. Убедитесь, что установлен флажок Explicit App ID (Явный идентификатор приложения), и полностью введите в стиле обратного доменного имени идентификатор пакета приложения, который вы хотите создать. Я задал здесь значение `com.pixolity.ios.cookbook.testingpasses`, а мой идентификатор типа талона (Pass Type ID) ранее был задан как `pass.pixolity.ios.cookbook.testingpasses`. Идентификатор типа талона может и не совпадать с идентификатором пакета, но если такое совпадение соблюдается, то в будущем вам будет гораздо проще находить нужный идентификатор типа талона, если уже известен идентификатор приложения.
8. В разделе App Services (Сервисы приложения) на этой странице нужно обязательно установить флажок Passbook. Так вы откроете в вашем приложении доступ к книге талонов Passbook.
9. Когда сделаете это, нажмите кнопку Continue (Продолжить). На следующем экране, который откроется перед вами (рис. 19.19), вы увидите все значения, которые ввели на предыдущей странице. Просмотрите их еще раз внимательно и, если вас все устроит, нажмите кнопку Submit (Отправить).
10. Теперь, когда вы активизировали использование талонов для данного App ID, создадим профиль инициализации. Перейдите в подраздел Provisioning Profiles (Профили инициализации) раздела iOS Provisioning Profile (Профиль инициализации iOS).
11. Мы собираемся создать профиль инициализации для разработки, а не Ad Hoc (Специальная сборка). Поэтому в подразделе Provisioning (Инициализация) раздела Development (Разработка) нажмите кнопку +.
12. Откроется следующий экран. На нем выберите элемент iOS App Development (Разработка приложения для iOS) и нажмите кнопку Continue (Продолжить).
13. Теперь вам будет предложено выбрать идентификатор приложения (App ID) для вашего профиля. Выберите идентификатор приложения, созданный вами раньше в этом разделе книги, и нажмите кнопку Continue (Продолжить) (рис. 19.20).
14. Теперь вы увидите список доступных сертификатов для разработки, которые есть у вас на портале. Выберите один или несколько сертификатов, с которыми хотите ассоциировать ваш профиль. Обычно принято ассоциировать профиль всего с одним сертификатом, но на портале, где трудится много программистов, у каждого из которых есть свой сертификат разработки, бывает необходимо создать такой профиль, который ассоциирован сразу с несколькими сертификатами. Сделав выбор, нажмите кнопку Continue (Продолжить).
15. Далее вы увидите список зарегистрированных устройств. Выберите из него те устройства, которые будут включены в ваш профиль. Сделав это, нажмите кнопку Continue (Продолжить).
16. На следующем экране система запросит у вас указать имя профиля. Дайте ему информативное имя, а затем нажмите кнопку Generate (Сгенерировать).


**Рис. 19.19.** Обзор нового идентификатора приложения (App ID) перед интеграцией его с Passbook

17. Когда ваш профиль будет сгенерирован, нажмите кнопку Download (Скачать), чтобы загрузить его на ваше устройство (рис. 19.21). После того как загрузка завершится, перетащите этот профиль в iTunes, чтобы установить его на вашем устройстве.

Существует несколько способов установки профиля инициализации в операционной системе OS X. Самый лучший и быстрый способ — просто перетащить профиль в iTunes. Профиль также можно установить, воспользовавшись Xcode. Какой бы способ вы ни выбрали, ни в коем случае не делайте двойного щелчка на профиле в ходе его установки. В случае двойного щелчка профиль установится на диске под совершенно невразумительным названием, и позже вам будет очень сложно найти нужный профиль среди множества других. Чтобы не засорять диск, пользуйтесь iTunes или Xcode для установки профилей инициализации. Все профили инициализации, установленные у вас на диске, можно просмотреть в файле ~/Library/MobileDevice/Provisioning Profiles/.

18. Теперь откройте ваш проект в Xcode. На вкладке Build Settings (Настройки сборки) выберите только что созданный профиль инициализации для отладочных сборок (Debug-only). То же самое можно сделать и для специальных сборок (Ad Hoc), но в схеме Release (Релиз) на вкладке Build Settings (Настройки сборки).


**Рис. 19.20.** Выбор правильного идентификатора приложения для нового профиля инициализации, создаваемого для целей разработки

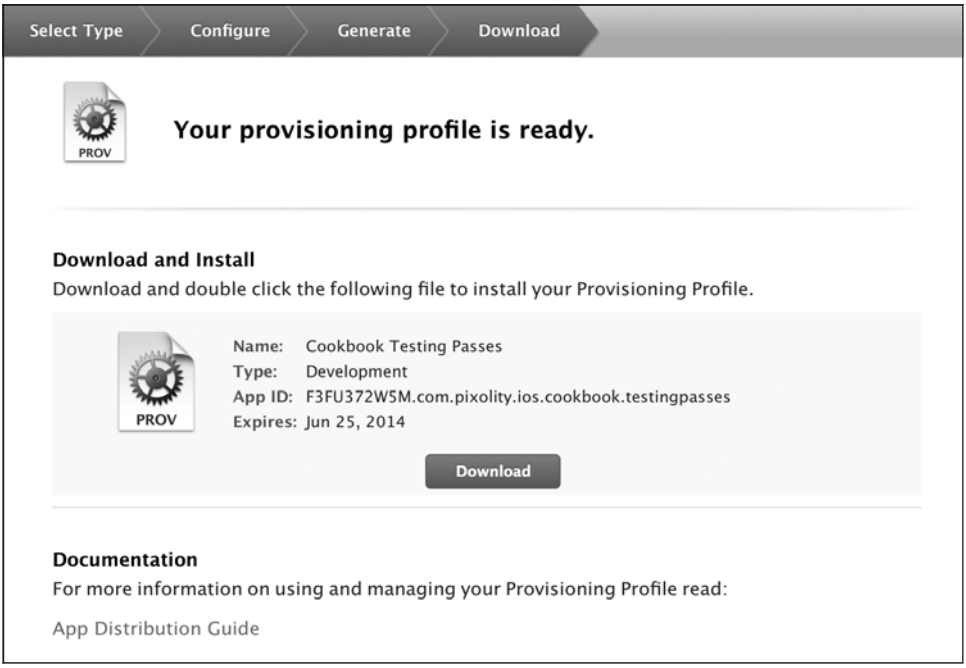


**Рис. 19.21.** Профиль Passbook готов к скачиванию

19. В Xcode рядом с вкладкой Build Settings (Настройки сборки) выберите Capabilities (Возможности), найдите там элемент Passbook и переведите его виртуальный переключатель в положение On (Включено) (рис. 19.22).

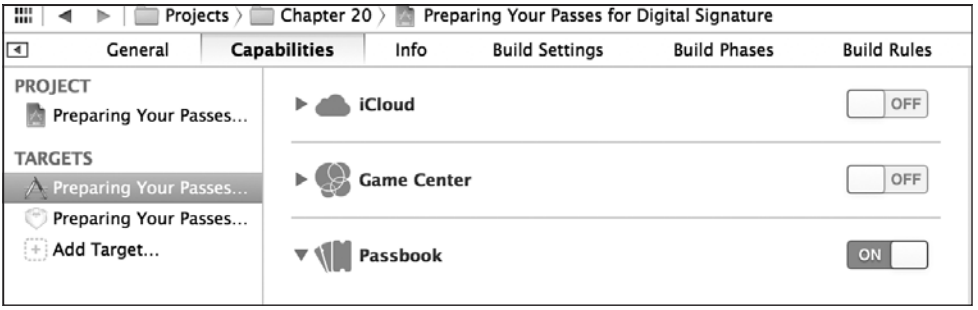


**Рис. 19.22.** Активизация Passbook в Xcode

20. Как только вы переведете переключатель для Passbook в положение On (Включено), Xcode свяжется с центром разработки и выберет оттуда все доступные для вас идентификаторы типа талона. В списке (рис. 19.23) выберите тот идентификатор типа талона, который вы создали ранее в этом разделе.

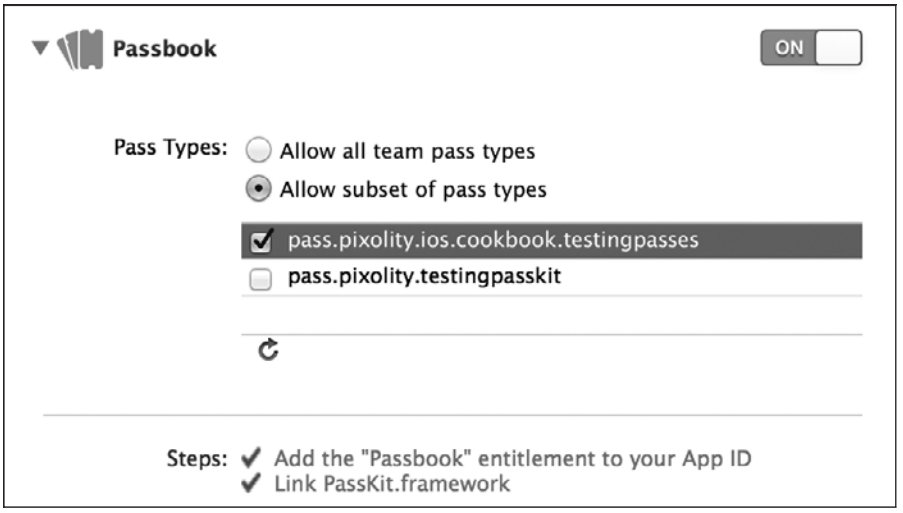


**Рис. 19.23.** Выбор корректного идентификатора типа талона в Xcode.

Вот мы и закончили настройку фреймворка Pass Kit! Осталось написать приложение, которое будет иметь доступ к талонам, расположенным на устройстве. Это приложение будет описано в разделе 19.9.

## См. также

Разделы 19.6 и 19.7.

# 19.9. Взаимодействие с Passbook с помощью программирования

## Постановка задачи

Требуется возможность программно взаимодействовать с талонами, установленными на пользовательском устройстве.

## Решение

Включите в проект PassKit.framework и воспользуйтесь библиотекой PKPassLibrary, чтобы найти интересующие вас талоны. Талоны относятся к типу `PKPass`. Используя этот класс, вы сможете получать информацию о талонах.

## Обсуждение

Для чтения этого раздела необходимо внимательно проработать раздел 19.8 и подготовить в Xcode проект для iOS, обладающий корректным профилем инициализации. Это нужно для доступа к вашим талонам, расположенным в пользовательской библиотеке Passbook.

Apple предоставила для iOS-разработчиков фреймворк PassKit.framework. Этот фреймворк позволяет взаимодействовать с талонами, которые пользователь установил на своем устройстве (устройствах). Чтобы можно было использовать этот фреймворк с применением новейшего компилятора LLVM, вам всего лишь потребуется импортировать в проект соответствующий обобщающий заголовок, вот так:

```
#import "AppDelegate.h"
#import <PassKit/PassKit.h>

<# Остаток вашего кода находится здесь #>
```

Далее потребуется объявить закрытое свойство типа `PKPassLibrary` в файле реализации делегата нашего приложения. Вышеупомянутый класс из фреймворка PassKit.framework позволяет взаимодействовать с талонами, добавленными на устройство. Для считывания значений, таких как номер платформы, с которой отправляется поезд, и город отправления, вам также нужно знать ключи из файла `pass.json`, который вы создали в разделе 19.2. Итак, объявим и эти ключи, тоже в файле реализации делегата приложения:

```
#import "AppDelegate.h"
#import <PassKit/PassKit.h>
@interface AppDelegate ()
@property (nonatomic, strong) PKPassLibrary *passLibrary;
@end

NSString *PassIdentifier = @"pass.pixolity.testingpasskit";
NSString *PassSerialNumber = @"p69f2J";

NSString *DepartureKey = @"departure";
```

```
NSString *DeparturePlatformKey = @"departurePlatform";
NSString *Arrival = @"arrival";
NSString *ArrivalPlatform = @"arrivalPlatform";

@implementation AppDelegate

<# Остаток вашего кода находится здесь #>
```

Великолепно! Написав этот код, вы теоретически получаете возможность доступа к библиотеке Passbook, расположенной на устройстве. Но погодите: а что делать, если на устройстве не установлена эта библиотека? Сначала нужно проверить, имеется ли библиотека Passbook на устройстве. Для этого используется метод класса `isPassLibraryAvailable`, относящийся к классу `PKPassLibrary`.

Далее нужно инстанцировать свойство `passLibrary` типа `PKPassLibrary`, а потом воспользоваться относящимся к библиотеке талонов методом экземпляра `passWithPassTypeIdentifier:serialNumber:`, чтобы найти искомый талон. Вот теперь понятно, почему среди различных ключей, относящихся к талону, мы, в частности, определяли идентификатор талона и его серийный номер. Вышеупомянутый метод вернет объект типа `PKPass`, который будет соответствовать вашему талону. Имея объект талона, вы можете считывать значения его ключей различными способами.

Ключи, задаваемые по умолчанию, в частности название организации и серийный номер, отображаются на свойства. Apple делает это за вас в классе `PKPass`. Однако если вы хотите получить доступ к значениям внутри `primaryFields` или в других подобных местах, то потребуется воспользоваться методом экземпляра `localizedValueForFieldKey:`, относящимся к классу `PKPass`. Мы сообщаем этому методу наши ключи, чтобы получить значения, ассоциированные с этими ключами. Далее показан небольшой фрагмент кода, позволяющий узнать информацию из талона, созданного в разделе 19.2: начальную и конечную точки маршрута, а также соответствующие железнодорожные платформы.

Код взят из файла реализации делегата нашего приложения.

```
#import "AppDelegate.h"
#import <PassKit/PassKit.h>

@interface AppDelegate ()
@property (nonatomic, strong) PKPassLibrary *passLibrary;
@end
NSString *PassIdentifier = @"pass.pixolity.testingpasskit";
NSString *PassSerialNumber = @"p69f2J";

NSString *DepartureKey = @"departure";
NSString *DeparturePlatformKey = @"departurePlatform";
NSString *Arrival = @"arrival";
NSString *ArrivalPlatform = @"arrivalPlatform";

@implementation AppDelegate

- (void) displayPassInformation:(PKPass *)paramPass{
```

```
  if (paramPass == nil){
    NSLog(@"The given pass is nil.");
    return;
  }

  NSLog(@"Departs From = %@",
  [paramPass localizedValueForFieldKey:DepartureKey]);
  NSLog(@"Departure Platform = %@",
  [paramPass localizedValueForFieldKey:DeparturePlatformKey]);
  NSLog(@"Arrives at = %@",
  [paramPass localizedValueForFieldKey:Arrival]);
  NSLog(@"Arrival Platform = %@",
  [paramPass localizedValueForFieldKey:ArrivalPlatform]);

}

- (BOOL) application:(UIApplication *)application
didFinishLaunchingWithOptions:(NSDictionary *)launchOptions{

  if ([PKPassLibrary isPassLibraryAvailable]){
    self.passLibrary = [[PKPassLibrary alloc] init];

    PKPass *pass =
    [self.passLibrary passWithPassTypeIdentifier:PassIdentifier
                                    serialNumber:PassSerialNumber];
    [self displayPassInformation:pass];

  } else {
    /* Здесь можно выполнить еще какое-нибудь действие */
    NSLog(@"The pass library is not available.");
  }

  self.window = [[UIWindow alloc]
                 initWithFrame:[[UIScreen mainScreen] bounds]];

  // Точка переопределения для дополнительной настройки после запуска
   приложения
  self.window.backgroundColor = [UIColor whiteColor];
  [self.window makeKeyAndVisible];
  return YES;
}
```

Идентификатор талона и серийный номер, указанные здесь, соответствуют талону, который я создал с помощью моего сертификата. Ваш серийный номер может быть таким же, но идентификатор талона определенно будет другим. Он должен быть информативен для вашей компании/на вашем портале инициализации.

## См. также

Раздел 19.2.