

*Разработка приложений
для iPhone на языке Objective-C*

**Второе
издание**

iPhone

Разработка приложений
с открытым кодом



O'REILLY®  bhv®

Джонатан Зdziарски

SECOND EDITION

iPhone Open Application Development

Jonathan Zdziarski

O'REILLY®

Beijing • Cambridge • Farnham • Köln • Sebastopol • Taipei • Tokyo

Джонатан Зdziарски

iPhone

Разработка приложений
с открытым кодом
2-е издание

Санкт-Петербург
«БХВ-Петербург»
2009

УДК 681.3.06
ББК 32.973.26-018.2
3-46

Здзиарски Дж.

3-46 iPhone. Разработка приложений с открытым кодом: Пер. с англ. —
2-е изд., перераб. и доп. — СПб.: БХВ-Петербург, 2009. — 368 с.: ил.

ISBN 978-5-9775-0397-6

Книга посвящена разработке приложений для iPhone на языке Objective-C с помощью iPhone API, используя последние версии инструментария с открытым кодом, обновленного для программного обеспечения iPhone 2.x и iPhone 3G. Рассматриваются настройка и работа с приложениями iPhone. Описана разработка пользовательских интерфейсов с помощью графической оболочки UIKit. Показана обработка событий. Рассмотрено программирование графики, включая анимацию и трехмерную трансформацию поверхностей. Уделено большое внимание вопросам записи и воспроизведения звуковых файлов. В приложении описаны различные приемы программирования и классы открытого кода для создания собственных приложений для iPhone.

Для программистов

УДК 681.3.06
ББК 32.973.26-018.2

Группа подготовки издания:

Главный редактор	<i>Екатерина Кондукова</i>
Зам. главного редактора	<i>Игорь Шишигин</i>
Зав. редакцией	<i>Григорий Добин</i>
Перевод с английского	<i>Александры Маленковой</i>
Редактор	<i>Анна Кузьмина</i>
Компьютерная верстка	<i>Натальи Смирновой</i>
Корректор	<i>Наталья Першакова</i>
Оформление обложки	<i>Елены Беляевой</i>
Зав. производством	<i>Николай Тверских</i>

Authorized translation of the English edition of iPhone Open Application Development, Second Edition, ISBN: 9780596155193, Copyright © 2009 Jonathan Zdziarski. All rights reserved. This translation is published and sold by permission of O'Reilly Media, Inc., the owner of all rights to publish and sell the same.

Авторизованный перевод английской редакции книги iPhone Open Application Development, Second Edition, ISBN: 9780596155193, © 2009 Jonathan Zdziarski. Все права защищены. Перевод опубликован и продается с разрешения O'Reilly Media, Inc., собственника всех прав на публикацию и продажу издания.

Лицензия ИД № 02429 от 24.07.00. Подписано в печать 27.02.09.
Формат 70×100^{1/16}. Печать офсетная. Усл. печ. л. 29,67.
Тираж 2000 экз. Заказ № 919
"БХВ-Петербург", 190005, Санкт-Петербург, Измайловский пр., 29.

Санитарно-эпидемиологическое заключение на продукцию № 77.99.60.953.Д.002108.02.07
от 28.02.2007 г. выдано Федеральной службой по надзору
в сфере защиты прав потребителей и благополучия человека.

Отпечатано с готовых диапозитивов
в ГУП "Типография "Наука"
199034, Санкт-Петербург, 9 линия, 12

ISBN 978-0-596-15519-3 (англ.)
ISBN 978-5-9775-0397-6 (рус.)

© 2009 Jonathan Zdziarski
© Перевод на русский язык "БХВ-Петербург", 2009

Оглавление

ПРЕДИСЛОВИЕ	1
Для кого предназначена эта книга.....	4
Структура книги.....	4
Используемые в этой книге обозначения.....	5
Использование примеров кода.....	5
Благодарности.....	6
ГЛАВА 1. ЗНАКОМСТВО С IPHONE И ЕГО НАСТРОЙКА	7
Процедуры взлома (jailbreak).....	8
Программное обеспечение для взлома от сторонних фирм.....	8
Установка SSH.....	10
Установка дополнительных компонентов UNIX.....	11
Дополнительные ресурсы.....	12
ГЛАВА 2. НАЧАЛО РАБОТЫ С ПРИЛОЖЕНИЯМИ	13
Анатомия приложения.....	13
Создание скелета приложения.....	15
Создание бесплатного пакета инструментов.....	18
Что вам потребуется.....	19
Компиляция пакета инструментов.....	21
Создание и установка приложений.....	24
Установка приложения.....	27
Переход к Objective-C.....	27
Сообщения.....	28
Объявление классов и методов.....	29
Реализация.....	31
Категории.....	32
Маскировка.....	35
ГЛАВА 3. ВВЕДЕНИЕ В UIKIT	37
Основные элементы пользовательского интерфейса.....	38
Окна и виды.....	40
Создание окна и вида.....	40
Отображение вида.....	41

Самое бесполезное приложение	42
Порождение от <i>UIView</i>	44
Второе самое бесполезное приложение	45
Текстовые виды	49
Создание текстового вида	49
Задание содержимого	50
Отображение текстового вида	50
Пример: отображение отказа от ответственности iPhone	50
Панели навигации	54
Создание панели навигации	55
Отображение панели навигации	58
Перехват нажатий кнопок	59
Запрещение кнопок	60
Добавление сегментного элемента управления	60
Пример: кнопка снижения громкости разговора с женой	61
Переходные виды	66
Создание перехода	67
Вызов перехода	67
Пример: переворачивание страниц	68
Листы действий	75
Создание листа действий	75
Кнопки листа действий	76
Отображение листа действий	77
Перехват нажатий кнопок	77
Отмена листа действий	78
Пример: кнопка "End-of-the-World"	78
Таблицы	84
Создание таблиц	84
Пример: проводник файлов	92
Манипуляции строкой состояния	102
Режим строки состояния	102
Скрытие строки состояния	104
Изображения строки состояния	105
Бейджи приложения	106
Отображение бейджа приложения	107
Удаление бейджа приложения	107
Сервисы приложения	108
Приостановка	108
Возобновление	110
Прекращение работы программы	111

ГЛАВА 4. ОБРАБОТКА СОБЫТИЙ И ПЛАТФОРМА GRAPHICS SERVICES	113
Введение в геометрические структуры	114
<i>CGPoint</i>	114
<i>CGSize</i>	115
<i>CGRect</i>	115
Введение в <i>GSEvent</i>	117
Graphics Services	117
События мыши	119
События жестов	122
События строки текущего состояния	124
Пример: перетаскивание значка	124
ГЛАВА 5. ГРАФИЧЕСКОЕ ПРОГРАММИРОВАНИЕ С ИСПОЛЬЗОВАНИЕМ CORE SURFACE И QUARTZ CORE ДЛЯ ОПЫТНЫХ ПОЛЬЗОВАТЕЛЕЙ.....	133
Уровни	134
Поверхности экрана	135
Создание поверхности экрана	135
Отображение поверхности экрана	136
Вывод на поверхность экрана	137
16-битные форматы пикселей	138
Буфер фрейма	139
Пример: случайный снег	139
Анимация уровня	144
Создание перехода уровней	144
Отображение перехода уровней	147
Пример: переворачивание страниц с применением стиля	148
Преобразования уровней	154
Пример: демонстрация вращения фонового рисунка	156
ГЛАВА 6. ЗВУК	163
Core Audio: великолепно, но вы не можете ее использовать	163
Celestial	164
Метод <i>ringerState</i>	165
Аудиоконтроллер	165
Аудиодорожки	169
Аудиоочереди	170
Пример: переменные мелодии звонка	172
Audio Toolbox	176
"Другая" аудиоочередь: для звука, генерируемого приложением	177

Пример: проигрыватель PCM	184
Запись звука	191
Пример: магнитофон	199
Уровень громкости	203
Пример: какой у меня уровень громкости?	206
ГЛАВА 7. ПРОЕКТИРОВАНИЕ В UIKit для ОПЫТНЫХ ПОЛЬЗОВАТЕЛЕЙ	211
Элементы управления	214
Сегментированные элементы управления	214
Переключающий элемент управления	218
Слайдеры	220
Таблицы предпочтений	222
Создание таблицы предпочтений	223
Отображение таблицы предпочтений	228
Пример: настройки игры-стрелялки	229
Индикаторы прогресса	239
<i>UIProgressIndicators</i> : то, что вертится	240
Пример: простой вращающийся индикатор	241
<i>UIProgressbar</i> : когда вращающиеся индикаторы не подходят	244
Пример: усовершенствованная строка прогресса	245
Progress HUDs: когда важно заблокировать любые действия	248
Пример: "Hello, HUD!"	249
Обработка изображений	252
Объект изображения	253
Пример: развлечение со значками	255
<i>UIImageView</i> : вид с видом	258
<i>UIAutocorrectImageView</i> : масштабирование	259
<i>UIClippedImageView</i> : обрезка кругов — квадраты	259
<i>UICompositeImageView</i> : многоуровневая прозрачность	260
Пример: интересная анимация обрезки	263
Списки разделов	267
Создание списка разделов	268
Отображение списка разделов	271
События выбора	271
Пример: выбор файлов	272
Выборщики	280
Создание выборщика	281
Отображение выборщика	283
Считывание выборщика	283
Пример: выбор типа вашего носа	283

Выборщик даты и времени	288
Создание выборщика даты и времени	288
Отображение выборщика даты	290
Считывание даты	290
Пример: независимый выборщик даты	291
Панели инструментов	294
Создание панели инструментов	294
Отображение панели инструментов	297
Бэйджи панели инструментов	297
Перехват нажатий кнопок	297
Пример: еще один подход к книге с текстом	297
Изменения ориентации	305
Считывание ориентации	306
Вращающиеся объекты	307
Пример: поворот мира в другую сторону	308
Считывание акселерометра	310
Виды Web-документа и прокрутки	311
Создание Web-вида	311
Как работают прокрутки	312
Использование класса <i>SimpleWebView</i>	317
Пример: простой обозреватель Интернета	318
ПРИЛОЖЕНИЕ. РАЗЛИЧНЫЕ ПРИЕМЫ И СПОСОБЫ	329
Выполнение дампа экрана	329
Пример: программа захвата экрана из командной строки	330
Выполнение дампа иерархии UI	332
Вызов Safari	333
Инициирование телефонных звонков	334
Вибрирование	334
Прозрачные виды	335
Переворачивание альбома в стиле Cover Flow	336
ПРЕДМЕТНЫЙ УКАЗАТЕЛЬ	347

Предисловие

Итак, вы хотите писать приложения для iPhone. Первое, что вы должны знать, — даже после выпуска первой версии SDK от Apple iPhone остается по большей части достаточно закрытым устройством, а Apple предпринимает шаги, чтобы не подпускать разработчиков близко к операционной системе. Тем не менее, это не удерживает людей от постижения всех глубин iPhone и создания приложений. С момента первой версии iPhone от Apple это популярное устройство прослыло самым трудновзламываемым мобильным устройством из когда-либо существовавших. Еще до конца 2007 года Web-узел <http://jailbreakme.com> Николаса Пенри (Nicholas Penree) сообщал, что более 30% рынка вломилось в их устройства, чтобы запускать несанкционированные приложения сторонних фирм на своих iPhone, с тех пор сообщество разработчиков неуклонно растет.

Команда хакеров iPhone, известная как iPhone Dev team, в течение первых 30 дней с момента выпуска iPhone 3G зафиксировала свыше шести миллионов посещений своего Web-узла. Большинство других узлов, хранящих средства взлома iPhone, имеют схожий рекордный по своей насыщенности трафик, и даже сотрудники Apple были пойманы на рекламе своих "настроенных" устройств.

В течение нескольких месяцев с момента выхода первой версии iPhone сообщество разработчиков открытого кода совершило геройский поступок, который привел к серьезному беспокойству Apple: это сообщество спроектировало бесплатный компилятор с открытым кодом и пакет инструментов для создания приложений iPhone, развернутые в специальном хранилище, и построило значительную базу для разработчиков. Все это было сделано для устройства, которое должно было быть закрытым. В 2008 году стало очевидно, что сообщество разработчиков программного обеспечения для iPhone не только существует, но и преуспевает в разработке выдающихся сторонних приложений для популярного устройства, и без какой-либо помощи от Apple. Чувствуя потенциально большой доход, Apple стремился реализовать собственную версию того, что сообщество разработчиков открытого кода уже создало: компилятор и пакет инструментов, специальное хранилище и большую базу для разработчиков. Можно только догадываться о том, явилось ли это результатом давления сообщества, но одно известно совершенно точно: со-

Предисловие

Итак, вы хотите писать приложения для iPhone. Первое, что вы должны знать, — даже после выпуска первой версии SDK от Apple iPhone остается по большей части достаточно закрытым устройством, а Apple предпринимает шаги, чтобы не подпускать разработчиков близко к операционной системе. Тем не менее, это не удерживает людей от постижения всех глубин iPhone и создания приложений. С момента первой версии iPhone от Apple это популярное устройство прослыло самым трудновзламываемым мобильным устройством из когда-либо существовавших. Еще до конца 2007 года Web-узел <http://jailbreakme.com> Николаса Пенри (Nicholas Penree) сообщал, что более 30% рынка вломилось в их устройства, чтобы запускать несанкционированные приложения сторонних фирм на своих iPhone, с тех пор сообщество разработчиков неуклонно растет.

Команда хакеров iPhone, известная как iPhone Dev team, в течение первых 30 дней с момента выпуска iPhone 3G зафиксировала свыше шести миллионов посещений своего Web-узла. Большинство других узлов, хранящих средства взлома iPhone, имеют схожий рекордный по своей насыщенности трафик, и даже сотрудники Apple были пойманы на рекламе своих "настроенных" устройств.

В течение нескольких месяцев с момента выхода первой версии iPhone сообщество разработчиков открытого кода совершило геройский поступок, который привел к серьезному беспокойству Apple: это сообщество спроектировало бесплатный компилятор с открытым кодом и пакет инструментов для создания приложений iPhone, развернутые в специальном хранилище, и построило значительную базу для разработчиков. Все это было сделано для устройства, которое должно было быть закрытым. В 2008 году стало очевидно, что сообщество разработчиков программного обеспечения для iPhone не только существует, но и преуспевает в разработке выдающихся сторонних приложений для популярного устройства, и без какой-либо помощи от Apple. Чувствуя потенциально большой доход, Apple стремился реализовать собственную версию того, что сообщество разработчиков открытого кода уже создало: компилятор и пакет инструментов, специальное хранилище и большую базу для разработчиков. Можно только догадываться о том, явилось ли это результатом давления сообщества, но одно известно совершенно точно: со-

общество разработчиков открытого кода доказало факт того, что все это можно вполне успешно реализовать.

В марте 2008 года был представлен Apple SDK. Этот SDK позволил разработчикам писать приложения с помощью санкционированных Apple инструментов и наградил возможностью продаж очень большой аудитории потенциальных покупателей через интернет-рынок Apple — AppStore. Однако многие разработчики посчитали такое предложение не очень интересным. Разрабатываемые таким образом приложения имеют уровень детских развлечений, нет возможности использовать многие компоненты операционной системы. Это не позволяло приложениям выполнять простые задачи, такие как работа в фоне, эффективное прекращение работы приложений отправки мгновенных сообщений и любых других служебных программ, работающих в фоне в режиме реального времени. Произвольные ограничения в Apple SDK также не давали разработчикам писать более сложные приложения, например, видеопроигрыватели, высококачественные 2D-игры и любые другие приложения, которые, по мнению Apple, можно было реализовать с помощью их собственного пакета программного обеспечения.

Помимо отсутствия возможности доступа к некоторым известным и ценным функциям iPhone, SDK ввел набор новых еще более ограничивающих объектов, которые "скрывали" от разработчика множество более низкоуровневых API. Хотя эти API и продолжали существовать на платформе iPhone (и использовались в действительности многими собственными приложениями Apple), ограничения этих интерфейсов давали, по мнению многих, нечестное преимущество Apple. Дополнительно к жалобам разработчиков добавилось соглашение Apple для разработчиков, раскритикованное за запрет создания определенных видов приложений, включая пошаговое навигационное программное обеспечение и байт-кодированные интерпретаторы (такие как Java и Flash).

Имея в виду все эти неоднозначные вопросы, многие разработчики решили примкнуть к рядам тех, кто пишет программное обеспечение с открытым кодом, не связанным с лицензионными и техническими ограничениями Apple, программное обеспечение которого действительно использует те же самые API, что и собственные приложения Apple, и способно получать доступ к функциям на существенно более низком уровне, нежели заурядное программное обеспечение AppStore. На сегодняшний день вы можете найти множество отличных приложений только в сообществе разработчиков открытого кода. Установщик программного обеспечения данного сообщества Cydia распространяет много различных приложений, например, игровые

эмуляторы, видеопроигрыватели, инструменты UNIX, менеджеры тем и другое великолепное программное обеспечение. Сообщество разработчиков открытого кода разработало собственные удобные инструменты, поскольку разработчики посчитали SDK от Apple чересчур технически и политически ограничивающим. Кроме того, множество отличных коммерческих пакетов вне рамок AppStore превосходят свои аналоги с AppStore просто потому, что они имеют ничем не ограниченный доступ к низкоуровневым API iPhone. Даже простейшие приложения карманного фонарика могут превосходить аналоги с AppStore всего лишь из-за возможности настройки яркости экрана. Наконец, пакет инструментов с открытым кодом имеет определенный уровень стабильности, который, в частности, предоставляет значительную совместимость кода между основными редакциями фирменного программного обеспечения — предмета большого разочарования разработчиков на SDK от Apple, которые стали жертвами частого переписывания кода.

Таким образом, имея под рукой пакет инструментов и множество бессонных ночей, сообщество изучило то, как использовать низкоуровневые платформы и интерфейсы на iPhone, чтобы создавать эффектные сторонние приложения, которые обладают потенциалом дать шанс AppStore заработать свои деньги. В данной книге мы рассмотрим платформы, являющиеся ключевыми в разработке полнофункционального программного обеспечения на iPhone, и обозначим те инструменты, которые могут воспользоваться преимуществами других недокументированных здесь платформ.

Хотя эта книга посвящена компилятору и инструментам сообщества разработчиков открытого кода, и не посвящена Apple SDK, тем не менее, разработчики, использующие SDK от Apple, обнаружат, что большая часть этой книги совпадает с объектами, напрямую или косвенно доступными в SDK, и что некоторые технические ограничения можно обойти, чтобы воспользоваться определенной функциональностью. Однако имейте в виду, что многие обсуждаемые здесь API официально запрещены для использования в SDK. Данная книга на самом деле предназначена для разработчиков открытого кода.

iPhone — великолепное устройство, и несмотря на политику, связанную с его недоступностью для разработчиков, сообщество его пользователей растет очень быстро. С помощью Apple или без нее iPhone дает рождение многим новым коммерческим рынкам и скоро превзойдет успех своих предшественников: PocketPC и Symbian, которые ранее владели рынком мобильных устройств.

По мере чтения этой книги вы, возможно, не будете осознавать то, как здорово, что она у вас есть. Простота, которую вы увидите в книге, является результатом тысяч часов работы активной части сообщества разработчиков, решающей практически невозможные задачи. Методы старой школы, использованные для работы с iPhone, были трудоемкими, если не сказать, изнурительными, и сами по себе могут составить тома книг.

Работа над открытием многих собственных интерфейсов на iPhone продолжается и сегодня. Мы приглашаем присоединиться к нам в нашей работе в сообществе всех, имеющих свои ноу-хау и огромное упорство в достижении поставленных целей.

Для кого предназначена эта книга

Чтобы эта книга была вам полезна, вы должны обладать определенными предварительными знаниями в программировании. Окружение iPhone использует Objective-C, с которым вы познакомитесь в *главе 2*. Положительным моментом является то, что вы в своих приложениях также можете использовать C и C++, поэтому все, кто имеет соответствующие познания, смогут быстро погрузиться в материал. Если вы не знаете C или C++, то данной тематике посвящена масса книг. Эта книга не является учебником для начинающих изучать эти языки, а раскрывает значение собственных классов и методов, необходимых для написания специализированных для iPhone приложений.

Структура книги

В *главе 1* объясняется, как проникнуть в ваш iPhone.

Глава 2 показывает состав приложения iPhone и то, как на вашем рабочем столе запускать пакет инструментов.

Глава 3 знакомит с UIKit, находящимся в центре разработки приложений iPhone и пользовательских интерфейсов.

Глава 4 описывает основные геометрические понятия, используемые в платформе Core Graphics, и уведомления о событиях.

Глава 5 погружается в процесс разработки для iPhone, предлагая изучить необработанные видеоповерхности и трансформации 3D.

Глава 6 посвящена множеству различных способов записи и воспроизведения звука и выходного потока цифрового аудио.

Глава 7 демонстрирует использование многих сложных компонентов пользовательского интерфейса из UIKit.

В *приложении* освещено несколько разнообразных приемов и классов открытого кода, позволяющих вам делать потрясающие вещи в ваших приложениях для iPhone.

Используемые в этой книге обозначения

В данной книге используются следующие соглашения об обозначениях:

- полужирный текст** применяется для обозначения названий меню, пунктов меню и кнопок меню, URL;
- курсив* используется для обозначения новых терминов;
- моноширинный шрифт служит для обозначения содержимого файлов, выходной информации команд, переменных, типов, классов, пространств имен, методов, объектов и всего того, что можно найти в программах.
- полужирный моноширинный шрифт** используется для обозначения команд или другого текста, которые должны точно вводиться пользователем, и частей кода или файлов, выделенных для обсуждения.

В книге также есть совет, примечание общего плана и предупреждения.

Использование примеров кода

Эта книга написана, чтобы помочь вам выполнить свою работу. Вообще говоря, вы можете использовать код, приведенный в этой книге, в ваших программах и документации. Вам не нужно связываться с нами, чтобы получить разрешение на использование кода, если только вы не собираетесь воспроизводить существенную часть кода. Например, для написания программы, использующей несколько фрагментов кода из этой книги, не требуется разрешения. Для продажи или распространения компакт-дисков с примерами из книг издательства O'Reilly обязательно требуется разрешение. Для ответа на вопрос с использованием цитат или выдержек из этой книги не требуется разрешения. Для включения значительного количества кода из примеров,

приведенных в данной книге, в документацию вашего продукта обязательно требуется разрешение.

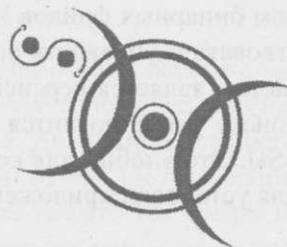
Мы будем признательны за ссылку на нашу книгу, хотя это и не обязательно. Ссылка обычно включает название книги, автора, издательство и ISBN. Например: "iPhone Open Application Development, Second Edition, by Jonathan Zdziarski. Copyright 2009 Jonathan Zdziarski, 978-0-596-15519-3".

Если вы считаете, что ваше использование примеров кода из этой книги выходит за рамки выданных выше разрешений, то не стесняйтесь связаться с нами по адресу: permissions@oreilly.com.

Благодарности

Выражаем особую благодарность Патрику Валтону (Patrick Walton), Джею Фриман (Jay Freeman), Брайану Вайтману (Brian Whitman), Джону Баффорду (John Bafford), Николасу Пенри (Nicholas Penree), Эллиоту Кроу (Elliot Kroo), Дино Пастосу (Dino Pastos), Нейт Тру (Nate True), Стиву Данхаму (Steve Dunham), Николасу Бакка (Nicolas Baccà), Даниэлю Пиблсу (Daniel Peebles), Александру Пику (Alexander Pick), Аарону Александру (Aaron Alexander), Ричарду Талли (Richard Thally), Джастину Лазарову (Justin Lazarow), Крису Зимману (Chris Zimman), Эрику Макдональду (Eric McDonald) и многим другим из сообщества разработчиков для iPhone, пожелавшим остаться анонимными, кто пожертвовал бессонными ночами и наличными из собственных карманов ради исследования iPhone и создания прочного основания для разработки приложений.

ГЛАВА 1



Знакомство с iPhone и его настройка

iPhone — достаточно закрытое устройство. Но нас это не устраивает. Программное обеспечение iPhone до версии 2.x включительно наглухо закрывало перед пользователями дверь в операционную систему, а разработчики вынуждены были довольствоваться игрой в кулички в строго ограниченной песочнице, созданной в пользовательском пространстве. Хотя эти ограничения не отпугивают большинство пользователей iPhone, но существенно затрудняют начало серьезной работы с ним. Прежде чем можно будет приступить к какому-либо доскональному изучению, iPhone, без преувеличения говоря, должен быть извлечен из своего заточения.

Взаимодействие iPhone с программным обеспечением, например, iTunes, происходит в chroot-окружении, в котором ни один пользователь и ни одно прикладное приложение — даже iTunes — не может видеть операционную систему; что в мире UNIX известно под названием chroot-тюрьмы (*chroot jail*). Эта тюрьма (и тот факт, что вы не можете просто вытащить жесткий диск) — единственная преграда, не позволяющая iPhone функционировать как полноценному переносному компьютеру Mac OS X. К счастью, существует множество бесплатных программных инструментов, упрощающих процесс извлечения iPhone из этого заточения.

В этой главе вы настроите ваш iPhone для разработки программного обеспечения таким образом, что сможете получать доступ к файлам вне этой тюрьмы, а ваши приложения смогут выполняться за рамками ограничивающей их песочницы. Для этого необходимо произвести освобождение из chroot-тюрьмы (называемое *jailbreaking*), тем самым вы получите доступ к файловой системе. Кроме того, вы установите BSD-мир UNIX, являющийся наборо-

ром бинарных файлов UNIX, таких как `ls` и `cp`. Это позволит вам взаимодействовать и управлять операционной системой iPhone, которая, как предполагается, является версией Mac OS X 10.5 (Leopard) для процессора ARM. Наконец, у вас появится безопасное командное окружение входа в систему, SSH. Это удобно для копирования файлов с вашего iPhone и на него, а также для установки приложений и выполнения примеров.

Процедуры взлома (jailbreak)

То, как вы будете освобождать свой iPhone из заточения (jailbreak), сильно зависит от используемой вами версии программного обеспечения. Существует временной разрыв в несколько недель, между выпуском нового программного обеспечения iPhone и появлением общедоступных инструментов для его взлома. Как правило, в новых версиях содержатся незначительные изменения, что каждый раз несколько затрудняет их взлом. Положительным моментом является то, что как только написан новый инструмент для взлома, то все свободно распространяемые программные средства обновляются, позволяя практически каждому осуществить данную процедуру.

Программное обеспечение для взлома от сторонних фирм

Существует множество бесплатных инструментов, предназначенных для взлома iPhone, одни более надежные, другие менее. Наилучшими средствами являются полнофункциональные программы, позволяющие вам помимо всего прочего легко и просто настраивать командный процессор и устанавливать программное обеспечение сторонних фирм. Перечислим лучшие инструменты.

- **iNdependenece**, <http://code.google.com/p/independence/> (версии 1.0.0—1.1.4).

iNdependenece — это программа для Mac OS X, осуществляющая взлом, активацию, установку SSH, и даже установку на iPhone звуков (ringtones), фоновых рисунков (wallpapers) и приложений сторонних фирм. *iNdependenece* работает под GPL, а автор создал библиотеку под названием *libPhoneInteraction*, позволяющую разработчикам писать другие инструменты для взаимодействия с iPhone.

- AppSnapp, <http://www.jailbreakme.com> (только версия 1.1.1).

Пользователи, применяющие встроенное программное обеспечение iPhone версии 1.1.1, могут посетить этот Web-узел с помощью своего iPhone и выполнить весь процесс взлома удаленно. Для проникновения в телефон AppSnapp пользуется уязвимостью в одной из библиотек изображений iPhone. Что особенно здорово на этом узле, так это то, что он не только взламывает ваш телефон, но еще и устраняет эту уязвимость, предотвращая тем самым возможность его злонамеренного использования. Кроме того, AppSnapp от версии 1.1.1 и далее исправляет программное обеспечение iPhone, чтобы разрешить выполнение приложений сторонних фирм, а также устанавливает AppTapp, установщик NullRiver, который затем может использоваться для создания условий в вашем iPhone для разработки программного обеспечения.

- AppTapp, <http://iphone.nullriver.com> (версии 1.0.0—1.0.2).

Nullriver — разработчик программного обеспечения из Онтарио (Канада), спроектировавший пакетный установщик для iPhone. Данный установщик позволяет вам с помощью нескольких касаний устанавливать на ваш iPhone любые приложения, имеющиеся в его хранилище. Само по себе программное обеспечение установщика работает с большинством версий программного обеспечения iPhone, но программа установки может взламывать встроенное программное обеспечение iPhone версии 1.0.x. Предыдущий инструмент в этом списке, AppSnapp, автоматически устанавливает AppTapp на устройства с версией 1.1.1. AppTapp также полезен для проведения процедуры уменьшения версии программного обеспечения, описанной далее.

- ZiPhone, <http://www.ziphone.org> (версии 1.0.0—1.1.4).

ZiPhone — это метод взлома, разработанный iPhone Dev Team. Он держался в величайшей тайне в преддверии выхода Apple SDK, но в итоге был выдан одним из бывших членов команды разработки. С тех пор ZiPhone был серьезно доработан и вышел за рамки всего лишь простого метода взлома, также к нему было добавлено множество других программ, включая полную разблокировку всех iPhone вплоть до OTB (Out-of-the-Box) версии 1.1.4.

- Pwnage, <http://www.iphone-dev.org> (версии 1.0.0—2.x).

Pwnage был первым инструментом, поддерживающим встроенное программное обеспечение версии 2.0 и iPhone 3G. Pwnage позволяет пользователю создавать собственный встроенный пакет, содержащий общий

установщик программного обеспечения под названием Cydia, и другие программные пакеты сторонних фирм. Pwnage пользуется слабыми местами iPhone и загрузочного ROM iPhone для загрузки в устройство неподписанного встроенного программного обеспечения. Для пользователей Windows существует Windows-версия Pwnage под названием WinPwn.

Установка SSH

После того как вы взломали ваш iPhone, установка Secure Shell позволит вам получить доступ к UNIX-окружению вашего iPhone и без труда копировать файлы на телефон и с него через беспроводное подключение WiFi.

Для использования SSH необходимо, чтобы ваш iPhone был подключен к той же беспроводной сети WiFi, что и ваш настольный компьютер. Если у вас нет доступа к беспроводной сети WiFi, то для установки приложений на ваш iPhone вам придется воспользоваться таким инструментом, как iNdependence, поэтому вы можете пропустить этот раздел. Однако вы можете попробовать установить MobileTerminal — бесплатную программу терминала для iPhone. По меньшей мере, это позволит вам работать в UNIX-окружении iPhone, что является необходимым условием для выполнения небольшого количества примеров. MobileTerminal может быть загружен прямо на iPhone с помощью Cydia или с адреса: <http://code.google.com/p/mobileterminal/>.

Независимо от инструмента, использованного вами для взлома вашего iPhone, общий установщик программного обеспечения Cydia должен быть добавлен на экран рабочего стола вашего iPhone. Чтобы установить SSH из Cydia:

1. Коснитесь значка Cydia, чтобы запустить само приложение. В начале Cydia может предложить вам обновить свое программное обеспечение. В этом случае пройдите процедуру обновления и перезагрузите установщик.
2. Коснитесь кнопки разделов, расположенной внизу, прокрутите сетевые пакеты (Networking packages), найдите и установите OpenSSH. Или же для его нахождения можете воспользоваться встроенной функцией поиска Cydia. Коснитесь кнопки **Install** и подтвердите установку OpenSSH.

Теперь на iPhone должен быть запущен SSH, но прежде чем вы сможете подключиться к нему, вам необходимо узнать IP-адрес вашего iPhone в вашей локальной беспроводной сети WiFi. Для этого:

1. На вашем iPhone коснитесь приложения **Settings**.

2. Выберите вкладку **General**, затем **Network**, затем **WiFi**.
3. В списке справа от вашей беспроводной сети WiFi должна быть отображена голубая стрелка.
4. Коснитесь голубой стрелки. Появится окно, содержащее ваш IP-адрес.

Чтобы упростить подключение, укажите ваш IP-адрес в файле host на вашем рабочем столе. Если вы пользуетесь Mac OS или UNIX, то можете отредактировать ваш файл /etc/hosts. Если вы пользуетесь Windows XP, то можете отредактировать файл C:\Windows\System32\drivers\etc\hosts. Добавьте в ваш файл следующую строчку:

```
x.x.x.x iphone
```

где x.x.x.x является IP-адресом вашего iPhone.

Теперь вы готовы подключиться к вашему iPhone с помощью клиента SSH. Если вы пользуетесь Mac OS или Linux с предустановленным SSH, то можете это сделать из терминального окна:

```
$ ssh -l root iphone
```

Если вы пользуетесь Windows XP, то вам придется загрузить клиента SSH. Наиболее распространенным бесплатным клиентом является PuTTY, доступный по адресу: <http://www.chiark.greenend.org.uk/~sgtatham/putty>.

В зависимости от того, какую версию программного обеспечения iPhone вы используете, заданным по умолчанию корневым паролем будет либо *dottie*, либо *alpine*. Как только вы войдете, то сразу же попадете в командную строку командного процессора.

Установка дополнительных компонентов UNIX

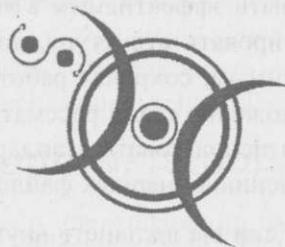
Сама по себе возможность доступа к командному процессору вашего iPhone ничего не дает без UNIX, предоставляющей основные команды. Приложение Cydia содержит базовую подсистему UNIX BSD Subsystem, но вам может потребоваться какой-либо конкретный инструмент, который не был установлен. Просмотрите список пакетов в Cydia и найдите те дополнительные инстру-

менты UNIX, которые вам требуются. Выберите и установите эти пакеты, коснувшись их, а затем коснувшись кнопки **Install**.

Дополнительные ресурсы

Apple периодически обновляет программное обеспечение iPhone, поэтому мы не можем описать то, как будет вести себя та или иная версия программного обеспечения — особенно новые версии, которые будут выпущены после публикации этой книги. Неоценимым ресурсом с точки зрения получения самой свежей информации о взломе вашего iPhone или установке приведенных в этой главе инструментов является следующий Web-узел команды разработчиков: iPhone Dev Team (<http://www.iphone-dev.org>). Это официальный узел команды разработчиков iPhone, отвечающей на сегодняшний день за большинство взломов версии 1.x и все известные версии 2.x.

ГЛАВА 2



Начало работы с приложениями

Если вы новичок в мире Mac, то будете удивлены тем фактом, что приложения не являются exe-файлами. Потрясающая архитектура аппаратного обеспечения и графики, которой славится Apple, распространяется и на архитектуру программного обеспечения, и на то, каким способом организованы приложения в файловой системе. Стратегия, используемая в настольных системах Apple, перенесена и на iPhone.

Apple практикует создание модульных независимых приложений с собственными внутренними файловыми ресурсами. В итоге процесс установки большинства приложений сводится всего лишь к простому перетаскиванию их в вашу папку приложений; а удаление их осуществляется путем перетаскивания их в корзину. В этой главе будет объяснена структура приложений iPhone. Кроме того, вы познакомитесь с набором бесплатных инструментов с открытым кодом, используемых для создания исполняемых файлов, а также узнаете, как устанавливать приложения на ваш iPhone. Наконец, вы познакомитесь с языком Objective-C, с его спецификой, достаточной для того, чтобы без труда перейти на него с C или C++.

Анатомия приложения

Apple использует элегантный способ содержать приложения в их операционных системах. Поскольку OS X является платформой на базе UNIX, то Apple хотелось, чтобы она придерживалась основных файловых соглашений UNIX, поэтому использование ветвей ресурсов стало недостаточным (перестало

быть эффективным в этих условиях). Суть задачи была в том, чтобы спроектировать структуру, которая бы позволила приложению оставаться независимым, сохраняя работоспособность в файловой системе. В результате приложение стали рассматривать как *пакет* (bundle) внутри *каталога* (directory) и использовать стандартные API для получения доступа к ресурсам, выполнению бинарных файлов и чтению информации о приложении.

Если вы взглянете внутрь любого приложения Mac, то обнаружите, что расширение app указывает не на файл, а на каталог. Это *программный каталог* (program directory). Внутри он является организованной структурой, содержащей ресурсы, которые необходимы приложению для выполнения, информацию о приложении и исполняемые бинарные файлы приложения. Такую структуру программного каталога компилятор не генерирует, а создает только исполняемые бинарные файлы. Поэтому, чтобы построить цельное приложение, разработчик сам создает скелет структуры, которая, в конечном счете, будет содержать бинарный файл и все его ресурсы.

Программный каталог для приложения iPhone гораздо менее структурирован, нежели настольные приложения Mac. На самом деле все файлы, используемые приложением, находятся в корне программной папки .app:

```
drwxr-xr-x root admin Terminal.app/  
  -rw-r--r-- root admin Default.png  
  -rw-r--r-- root admin Info.plist  
  -rwxr-xr-x root admin Terminal  
  -rw-r--r-- root admin icon.png  
  -rw-r--r-- root admin pie.png
```

Приведенный выше пример отражает самое основное приложение iPhone под названием MobileTerminal. MobileTerminal — это терминальный клиент с открытым кодом для iPhone, позволяющий пользователю улучшить командный процессор и работать в окружении UNIX (которое тоже должно быть установлено как программное обеспечение сторонних фирм). MobileTerminal иллюстрирует все основные компоненты приложения iPhone:

- MobileTerminal.app — каталог, в котором находятся все ресурсы приложения;
- Default.png — изображение в формате PNG (Portable Network Graphics). Когда пользователь запускает приложение, iPhone анимирует его, чтобы показать, как оно переходит на передний план экрана. Это делается с помощью загрузки файла Default.png и *масштабирования* его до размера всего экрана. Это изображение размером 320×480 пикселей перемещается

на передний план и остается на экране до тех пор, пока приложение не закончит свою загрузку, после чего оно становится фоновым рисунком для любых элементов пользовательского интерфейса, отображаемых на экране. Как правило, для фонового рисунка приложения используют сплошной черный или белый цвета;

- `Info.plist` — список свойств, содержащий информацию о приложении. Он содержит название исполняемого бинарного файла и идентификатор пакета, используемый приложением SpringBoard для его загрузки. Позднее в этом разделе будет приведен пример списка свойств;
- `Terminal` — фактически исполняемый бинарный файл, вызываемый при запуске приложения. Это то, что выдает ваш компилятор при построении приложения. При создании конечной производственной версии ваш сборочный файл проекта (`make-файл`) может копировать ваш бинарный файл в папку приложения. В этой главе будет представлен пример такого процесса;
- `icon.png` — изображение, создающее значок приложения в SpringBoard (приложение рабочего стола iPhone). SpringBoard не волнует размер файла, и оно будет пытаться отобразить изображение за пределами области значка, если то достаточно велико. Большинство значков, как правило, имеет размер 60×60 пикселей;
- `pie.png` — изображение, используемое приложением MobileTerminal. Окружение iPhone предоставляет множество методов для выбора ресурсов, большинство из них принимают только имя файла и не принимают путь к файлу. Поэтому файл, передаваемый в эти методы, должен храниться непосредственно в программном каталоге. Такой подход соответствует попыткам Apple делать приложения независимыми.

Создание скелета приложения

Первое, что вы должны сделать прежде, чем приступить к созданию приложения, — это составить скелет каталога `.app`, который будет его содержать.¹ Скелет будет предоставлять всю информацию, необходимую iPhone для под-

¹ Технически вполне возможно запускать приложение напрямую из командной строки iPhone, но это рушит многие функции уровня приложения. Само по себе SpringBoard тесно интегрировано с платформой пользовательского интерфейса, поэтому чтобы ваше приложение стало полностью работоспособным, оно должно быть соответствующим образом смонтировано и вызвано.

тверждения существования вашего приложения, что позволит запускать его из SpringBoard.

В этой книге представлено множество полнофункциональных примеров кода. Для корректного запуска их вы должны построить пример скелета под названием MyExample.app. Создать каталог достаточно просто:

```
$ mkdir MyExample.app
```

Далее, напишите список свойств, чтобы описать само приложение и то, как его запустить. Файл Info.plist отображает список свойств в формате XML. Он должен выглядеть следующим образом:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple Computer//DTD PLIST 1.0//EN"
"http://www.apple.com/DTDs/PropertyList-1.0.dtd">
<plist version="1.0">
<dict>
  <key>CFBundleDevelopmentRegion</key>
  <string>English</string>
  <key>CFBundleExecutable</key>
  <string>MyExample</string>
  <key>CFBundleIdentifier</key>
  <string>com.oreilly.www.iphone.examples</string>
  <key>CFBundleInfoDictionaryVersion</key>
  <string>6.0</string>
  <key>CFBundlePackageType</key>
  <string>APPL</string>
  <key>CFBundleSignature</key>
  <string>????</string>
  <key>CFBundleVersion</key>
  <string>1.0</string>
</dict>
</plist>
```

В приведенном примере большинство наиболее важных параметров выделено полужирным шрифтом. Это значения свойств CFBundleExecutable и CFBundleIdentifier. Свойство CFBundleExecutable задает имя исполняемого бинарного файла в пределах папки. Это именно тот файл, который начинает исполняться при запуске вашего приложения, т. е. то, что выдает ваш

компилятор. В данном примере имя файла совпадает с именем приложения, но это совершенно необязательно.

Свойство `CFBundleIdentifier` определяет уникальный идентификатор, под которым будет находиться ваше приложение. Уровень приложения iPhone больше заботится об адресации вашего приложения целиком, нежели самого бинарного файла. Всякий раз, когда SpringBoard (или другое приложение) запускает приложение `MyExample`, сослаться на него нужно будет с использованием этого идентификатора. Его имя должно быть уникальным среди всех других приложений на iPhone. Наиболее распространенным способом, гарантирующим уникальность идентификатора, является включение в его имя URL вашего Web-узла.

Файлы приложения `icom.png` и `Default.png` тоже должны быть скопированы. Если это будет упущено, то iPhone использует вместо них по умолчанию совершенно неприглядные изображения. Чтобы показать, что имеется в виду, мы не будем включать эти файлы в наш пример. Но чтобы ваше приложение выглядело вполне профессионально, убедитесь, что при публикации своего приложения вы создали изображения и включили файлы этих изображений с такими именами.

Теперь наш скелет вполне подходит для выполнения примера. В следующем разделе вы установите пакет инструментов на ваш рабочий стол, после чего сможете компилировать примеры приложений. В последующих главах вы создадите множество примеров. После того как каждый из них будет построен, исполняемый бинарный файл `MyExample` нужно будет скопировать в вашу программную папку. Законченное ваше приложение будет выглядеть следующим образом:

```
drwxr-xr-x  root  admin  MyExample.app/
-rw-r--r--  root  admin  Info.plist
-rwxr-xr-x  root  admin  MyExample
```

Примеры, приводимые в этой книге, как правило, не требуют никаких дополнительных ресурсов, поэтому изображения и звуки будут необходимы только при вызове их из примеров. Однако в этих случаях вы скопируете требуемые файлы в каталог `MyExample.app`. Большинство примеров старается использовать существующие на iPhone файлы, чтобы не переполнять эту книгу большим количеством бинарного кода.

Создание бесплатного пакета инструментов

Как мы уже говорили в *главе 1*, iPhone зарождался как закрытая платформа. Изначально это означало, что никакой инструментарий разработчика для создания приложений для iPhone не будет общедоступен. Было много разговоров о том, что Apple втайне надеялась на то, что сообщество взломает телефон, подтверждая тем самым свой статус среди помешанных на этом сообществ. В течение первых нескольких месяцев жизни iPhone именно это и происходило. Сообщество открытого кода успешно взломало телефон и начало написание пакета инструментов для создания приложений. С тех пор оно уже выпущено как бесплатное программное обеспечение. Пакет инструментов состоит из кросс-компилятора, компоновщика, транслятора, блока C в компоновщике под названием Csu и заголовков классов для платформы Objective-C, генерируемых инструментом под названием class-dump.

К сегодняшнему дню данный пакет инструментов претерпел множество изменений и усовершенствований, сделанных Джейм Фриманом, также известным как saurik, и доступен в двух видах. Настольная версия пакета инструментов использует кросс-компилятор, выполняющийся на одной машине (а именно на вашей настольной), но создает исполняемые файлы, которые могут исполняться на другой машине (в iPhone это процессор ARM). Родной компилятор также может быть установлен напрямую на iPhone, позволяя тем самым вам создавать приложения без необходимости установки на ваш рабочий стол какого-либо специализированного программного обеспечения.

Приводя в этой книге команды и пути к файлам, мы исходим из того, что для создания и/или установки данного пакета инструментов вы использовали процедуры из этой главы. Обновление пакета инструментов происходит периодически в виде выпуска новых версий, поэтому процесс его установки может иногда меняться. Самые последние инструкции по созданию пакета инструментов на рабочем столе можно найти на Web-узле Джея Фримана по адресу: <http://www.saurik.com>.

По умолчанию пакет инструментов создается и устанавливается в папку /toolchain. Все приводимые в этой книге примеры предполагают, что именно в нее был установлен пакет. Если же вы создали пакет инструментов ранее или же только что обеспокоились модификацией файлов в ней, то вы наверняка захотите переместить вашу текущую папку /toolchain в другое место и начать с чистого каталога.

ВНИМАНИЕ!

Если у вас имеется предыдущая версия пакета инструментов, то более новая версия может создаться некорректно. Чтобы убедиться в том, что вы начинаете с чистой установки, переместите вашу старую копию в другое место.

Что вам потребуется

Если вы ищете способ установки пакета инструментов напрямую в iPhone, то единственное, что вам потребуется, — это взломанный iPhone с запущенным на нем установщиком Cydia. Чтобы установить пакет инструментов, просто запустите Cydia, а затем в списке **Sections** выберите раздел **Development**. Найдите и выделите в списке iPhone 2.0 Toolchain, затем коснитесь кнопки **Install**. На ваш iPhone будет установлен пакет инструментов, и вы сможете вызывать его с помощью стандартной команды `gcc`. Теперь вы можете пропустить оставшуюся часть данного раздела.

Создание пакета на рабочем столе — это более сложный и запутанный процесс. Хотя в Интернете существует несколько неофициальных бинарных дистрибутивов пакета инструментов, вы создадите его из источников, приводимых в этом разделе.

Поддерживаемые настольные платформы

Первое, что вам потребуется, — это поддерживаемая настольная платформа. На текущий момент пакетом инструментов поддерживаются следующие платформы:

- Mac OS X 10.4 Intel или PPC;
- Mac OS X 10.5 Intel;
- Ubuntu Feisty Fawn, Intel;
- Ubuntu Gutsy Gibbon, Intel;
- Fedora Core, Intel;
- Gentoo Linux 2007.0, x86_64;
- Debian 2.6.18;
- CentOS 4.

Другие платформы следуют тем же основным шагам, что и приведенные выше. Официальные инструкции пакета инструментов можно найти на Web-узле <http://www.saurik.com>.

Высокоскоростное подключение к Интернету

Пакет инструментов имеет достаточно большой объем, даже в виде исходных файлов. Если только вы не хотите провести в ожидании несколько дней, то вы, скорее всего, загрузите исходные файлы по высокоскоростному каналу Интернета. Если у вас нет такого канала, то лучше произвести установку из библиотеки или в местном интернет-кафе.

Инструменты с открытым кодом

Следующее, что вам потребуется, — это набор обязательных инструментов с открытым кодом, установленных на вашем рабочем столе:

- bison (версия 1.28 или позднее);
- flex (версия 2.5.4 или позднее);
- gcc (компилятор GNU, который обрабатывает команды C, C++ и Objective-C);
- git (утилита контроля исходного кода).

Если у вас нет каких-либо из этих инструментов, то прежде чем продолжить, загрузите и установите их. На Mac все эти инструменты, кроме git, входят в состав набора инструментов Xcode, но прежде чем продолжить, лучше установить или обновить Xcode на самую последнюю версию. Большинство других операционных систем предлагает эти инструменты в своих дистрибутивах как необязательные компоненты.

ПРИМЕЧАНИЕ

Инструменты Xcode могут быть загружены с Web-узла Apple по адресу: <http://developer.apple.com/tools/xcode>.

Файловая система iPhone

Вам необходимо сделать копию файловой системы вашего iPhone, особенно библиотек и оболочек. Из-за возможных необратимых последствий и в связи

с тем, что наши юристы заставляют нас сделать это, мы публикуем этот отказ от ответственности:

ВНИМАНИЕ!

Установка пакета инструментов требует от вас создания копии библиотек с вашего iPhone на ваш рабочий стол. Убедитесь в том, что это не противоречит местному и федеральному законодательству, а также законодательству штата, в котором вы находитесь.

После установки SSH на iPhone (см. главу 1) для загрузки в папку /toolchain/sys необходимых вам файлов воспользуйтесь следующими командами:

```
$ sudo -s
# mkdir -p /toolchain/sys/
# cd /toolchain/sys/
# mkdir -p ./System/Library ./usr
# scp -r root@iphone:/System/Library/Frameworks/ ./System/Library
# scp -r root@iphone:/System/Library/PrivateFrameworks/ ./System/Library
# scp -r root@iphone:/usr/lib ./usr
```

Компиляция пакета инструментов

Прежде чем вы начнете, вы должны задать несколько переменных окружения, чтобы определить, куда устанавливается пакет инструментов. Чтобы установить в /toolchain, используйте приведенные далее пути. Конечно, вы можете изменить их согласно своим предпочтениям. Убедитесь в том, что вы поместили загруженные библиотеки и оболочки туда, куда указано в \${sysroot}:

```
# export target=arm-apple-darwin9
# export prefix=/toolchain/pre
# export sysroot=/toolchain/sys
# export PATH="${prefix}/bin":$PATH
# export cctools=/toolchain/src/cctools
# export gcc=/toolchain/src/gcc
# export csu=/toolchain/src/csu
# export build=/toolchain/bld
```

Как только все переменные будут экспортированы, вы готовы к созданию пакета.

Шаг 1. Установка Csu

Csu предоставляет C-вход (C hooks) в точку входа "start" сборки и задает стек, чтобы могла быть вызвана функция `main()` вашей программы. Это, по сути, связующий код:

```
# mkdir -p ${csu}
# cd "${csu}"
# svn co http://iphone-dev.googlecode.com/svn/trunk/csu .
# cp -R *.o "${sysroot}"/usr/lib
# cd "${sysroot}"/usr/lib
# chmod 644 *.o
# cp -Rf crt1.o crt1.10.5.o
# cp -Rf dylib1.o dylib1.10.5.o
```

Шаг 2. Построение и установка инструментов кросс-компилятора

Приведенные далее команды создают и устанавливают компоненты кросс-компилятора пакета инструментов.¹ Это весьма специфично для Mac OS X, поэтому если вы используете другую платформу, то обратитесь к ее официальной документации:

```
# rm -rf "${cctools}"
# svn co http://iphone-dev.googlecode.com/svn/branches/odccctools-9.2-1d "${cctools}"
# mkdir -p "${build}"
# cd "${build}"
# mkdir cctools-iphone
# cd cctools-iphone
# CFLAGS=-m32 LDFLAGS=-m32 "${cctools}"/configure \
  --target="${target}" \
```

¹ Здесь и далее в коде встречается символ `\`, означающий перенос строки. То есть код между этим символом на самом деле надо набирать в одну строку. — *Ред.*

```
--prefix="${prefix}" ↵
--disable-ld64
# make
# make install
```

Шаг 3. Установка системных заголовков

Системные заголовки, имеющиеся в вашем Xcode SDK, являются общими для вашего рабочего стола и платформы iPhone, однако некоторые макросы прекомпилятора сильно привязаны к архитектуре. Поскольку архитектура iPhone отличается от архитектуры рабочего стола, то для того чтобы эти заголовки работали для iPhone, их необходимо установить. Чтобы установить специфичный для iPhone набор заголовков в ваш недавно созданный пакет инструментов, выполните приведенные далее команды. Все это основано на заголовках Xcode:

```
# cd "${build}"
# svn co http://iphone-dev.googlecode.com/svn/branches/include-1.2-sdk
include
# cd include
# ./configure --prefix="${sysroot}"/usr
# bash install-headers.sh
```

Шаг 4. Построение и установка LLVM

После установки заголовков последним шагом является создание компилятора LLVM. Оболочка LLVM (Low Level Virtual Machine) обеспечивает стандартную инфраструктуру для создания компиляторов. Она предоставляет необходимые входы и API для создания стандартизованного компилятора без необходимости переписывать все основные компоненты компилятора. Чтобы скомпилировать и установить версию компилятора LLVM, выполните следующие команды:

```
# rm -rf "${gcc}"
# git clone git://git.saurik.com/llvm-gcc-4.2 "${gcc}"
# mkdir -p "${build}"
# cd "${build}"
# mkdir gcc-4.2-iphone
# cd gcc-4.2-iphone
```

```
# "${gcc}"/configure ↵
  --target="${target}" ↵
  --prefix="${prefix}" ↵
  --with-sysroot="${sysroot}" ↵
  --enable-languages=c,c++,objc,obj-c++ ↵
  --with-as="${prefix}/bin/${target}-as" ↵
  --with-ld="${prefix}/bin/${target}-ld" ↵
  --enable-wchar_t=no ↵
  --with-gxx-include-dir=/usr/include/c++/4.0.0
# make -j2
# make install
# mkdir -p "${sysroot}"/"${dirname "${prefix}"}"
# ln -s "${prefix}" "${sysroot}"/"${dirname "${prefix}"}"
```

Создание и установка приложений

Теперь, когда установлен пакет инструментов, пришло время узнать, как им пользоваться. Существуют два основных способа создания исполняемых файлов: командная строка и make-файл.

Приводимые в этой книге примеры достаточно просты, поэтому они могут быть созданы с помощью командной строки. Пакет инструментов совместим со стандартными аргументами компилятора и должен быть вам знаком, если вы когда-либо пользовались gcc. Прежде чем попытаться воспользоваться кросс-компилятором, убедитесь в том, что среди ваших путей имеется /toolchain/pre/bin:

```
$ export PATH=$PATH:/toolchain/pre/bin
```

Анатомия типичного компилятора командной строки следующая:

```
$ arm-apple-darwin9-gcc -o MyExample MyExample.m -lobjc ↵
  -framework CoreFoundation -framework Foundation ↵
  -march=armv6 -mcpu=arm1176jzf-s
```

Здесь:

- arm-apple-darwin9-gcc — само название кросс-компилятора. Оно располагается в /toolchain/pre/bin, поэтому убедитесь в том, что эта папка добавлена в ваши пути;

- `-o MyExample` — сообщает компилятору выводить скомпилированный исполняемый файл в файл с названием `MyExample`;
- `MyExample.m` — имена исходных файлов, включенных в программу, разделяемых пробелами. Расширения `m` сообщает компилятору о том, что исходники написаны на Objective-C;
- `-lobjc` — указывает компилятору компоновать библиотеки сообщений Objective-C, необходимые всем приложениям iPhone, в пакет инструментов. Помимо всего прочего, эта библиотека склеивает вызовы функций C-стиля с сообщениями Objective-C.
- `-framework CoreFoundation -framework Foundation` — компоновка двух основных оболочек в приложение. В зависимости от того, какие компоненты операционной системы используются в коде, разные оболочки предоставляют различные уровни функциональности. В этой книге вы познакомитесь со множеством различных оболочек;
- `-march=armv6 -mcpu=arm1176jzf-s` — задает корректную архитектуру и тип процессора исполняемого файла.

Командной строки будет более чем достаточно для большинства небольших приложений и примеров, но для больших приложений имеет смысл писать *make-файл* (makefile). Make-файл — это простой текстовый файл, который выступает в роли декларации при построении приложений. Он используется программой `make`, являющейся портативной утилитой построения, поставляемой с множеством наборов средств разработки. Программа `make` отвечает за вызов компилятора (и компоновщика) и передачу им необходимых флагов и параметров. Make-файлы являются логическими способами спланировать структуру приложения. Кроме того, они позволяют разработчику легко наводить порядок среди объектных файлов в каталоге, создавать пакеты приложений, а также выполнять целый ряд других задач, весьма полезных при создании приложений.

Предыдущий пример командной строки может быть переписан в make-файл так, как показано далее. Make-файл назван `Makefile` и помещен в исходный каталог:

```
CC = /toolchain/pre/bin/arm-apple-darwin9-gcc
LD = $(CC)
LDFLAGS = -lobjc \
          -framework CoreFoundation \
          -framework Foundation
CFLAGS = -march=armv6 -mcpu=arm1176jzf-a
```

```
all:      MyExample

MyExample: MyExample.o
           $(LD) $(LDFLAGS) -o $@ $^

%.o:     %.m
           $(CC) -c $(CFLAGS) $(CPPFLAGS) $< -o $@

%.o:     %.c
           $(CC) -c $(CFLAGS) $(CPPFLAGS) $< -o $@

%.o:     %.cpp
           $(CC) -c $(CFLAGS) $(CPPFLAGS) $< -o $@
```

ВНИМАНИЕ!

Все отступы являются в действительности табуляциями. Табуляции необходимо использовать для корректной работы make-файла.

Как только make-файл создан, исполняемый файл приложения может быть создан с помощью всего одной простой команды:

```
$ make
```

Кроме создания приложения можно добавить функциональность для копирования исполняемого файла приложения в созданный вами скелет программной папки:

```
package:
           cp -p MyExample ./MyExample.app/
```

Добавив это в make-файл, вы сможете запустить `make package` для автоматического задания вашего каталога `.app`.

Другой весьма распространенный вариант использования make-файлов — это очистка каталога, чтобы его можно было отправить кому-либо другому. Вы даже можете указать make-файлу удалить исполняемый файл, который был скопирован в программную папку:

```
clean:
           rm -f *.o *.gch
           rm -f ./MyExample.app/MyExample
```

Установка приложения

После того как приложение будет создано, оно может быть установлено путем копирования всего программного каталога в папку /Applications на iPhone. С помощью сервера SSH, который вы настроили в *главе 1*, это можно сделать через WiFi:

```
$ scp -r MyExample.app root@iphone:/Applications
```

Чтобы iPhone обнаружил приложение, должен быть выключен и перезагружен либо сам iPhone, либо же на самом iPhone должно быть перезапущено приложение SpringBoard. Войдите в iPhone с помощью SSH и для перезапуска SpringBoard выполните следующую команду:

```
$ killall SpringBoard
```

После этого вы увидите ваше приложение в SpringBoard.

Если ваш iPhone управляется встроенным программным обеспечением версии 2.0 или старше, то для того, чтобы ваше приложение запускалось, вам необходимо выдать ему цифровую подпись. Это может быть сделано прямо на iPhone путем установки инструмента идентификации связей (link-identity). Это делается с помощью Cydia, или же, если у вас запущен SSH, то вы можете использовать инструмент apt-get из командной строки:

```
# apt-get install ldid
```

После того как вы установите ldid, подпишите ваше приложение с помощью следующей команды:

```
# ldid -S /Applications/MyExample.app/MyExample
```

Теперь вы можете коснуться значка приложения в SpringBoard, чтобы запустить его.

Переход к Objective-C

Objective-C был написан в начале 80-х годов прошлого века ученым и разработчиком программного обеспечения Брэдом Коксом (Brad Cox). Он разрабатывался как способ введения возможностей языка Smalltalk в программное окружение C. Большая часть библиотек оболочки iPhone написана на Objective-C, но поскольку этот язык проектировался так, чтобы обеспечить совместимость с языком C, то вы вполне свободно можете применять в ва-

шем приложении C и C++. Objective-C в основном используется в Mac OS X и GNUStep (бесплатная оболочка OpenStep). Некоторые языки, например Java и C#, многое позаимствовали из языка Objective-C. Оболочка Cocoa интенсивно использует Objective-C на рабочих станциях Mac, что и было перенесено в iPhone.

Если вы раньше разрабатывали программное обеспечение на Mac OS X, то вы уже знакомы с Objective-C, но если же iPhone стал вашей первой платформой Apple, тогда вы, скорее всего, переходите с C или C++. В этом разделе будут освещены некоторые наиболее существенные различия между этими языками. Если у вас есть опыт программирования на C или C++, то этого должно быть вполне достаточно для того, чтобы начать писать код, используя примеры из этой книги в качестве руководства.

Сообщения

Первое, что вы заметите в Objective-C, — это активное использование скобок. В Objective-C методы не *вызываются* (called) в обычном понимании этого действия, им отправляются *сообщения* (messages). Аналогично, метод не *возвращает* (return), а вместо этого *отвечает* (responds) на сообщение. В отличие от C, где вызовы функций должны быть предопределены, такой стиль Objective-C обмена сообщениями позволяет разработчику динамически создавать новые методы и сообщения прямо во время исполнения. Обратной стороной этого является то, что становится вполне вероятной отправка объекту такого сообщения, на которое он не сможет ответить, что в результате приведет к исключению и, скорее всего, к аварийному завершению программы.

Имея объект `myWidget`, сообщение его методу `powerOn` может быть отправлено следующим способом:

```
returnValue = [ myWidget powerOn ];
```

Аналог этого действия на C++ может выглядеть так:

```
returnValue = myWidget->powerOn();
```

Аналог на C может объявить функцию внутри его плоской системы имен:

```
returnValue = widget_powerOn(myWidget);
```

C сообщениями также могут передаваться аргументы в предположении, что объект может получать их.

Приведенный далее пример вызывает метод `setSpeed` и передает два аргумента:

```
returnValue = [ myWidget setSpeed: 10.0 withMass: 33.0 ];
```

Обратите внимание, что аргумент явно назван в сообщении. Это позволяет использовать множество методов с одинаковыми именами и типы данных для объявления — полиморфизм на стероидах:

```
returnValue = [ myWidget setSpeed: 10.0 withMass: 33.0 ];
```

```
returnValue = [ myWidget setSpeed: 10.0 withGyroscope: 10.0 ];
```

Объявление классов и методов

Несмотря на то, что классы C++ могут быть определены в Objective-C, основной причиной применения данного языка является возможность использования собственных объектов и средств Objective-C. Это относится и к использованию им интерфейсов. В стандартном C++ классы являются структурами, а их переменные и методы заключены внутри этих структур. С другой стороны, Objective-C хранит свои переменные в одной части класса, а методы — в другой. Этот язык также требует того, чтобы объявление интерфейсов осуществлялось специальным образом в собственном блоке кода (под названием `@interface`) отдельно от блока, содержащего реализации (под названием `@implementation`). Сами методы также построены в манере Smalltalk и очень слабо походят на обычные функции C.

Интерфейс для нашего простенького примера может выглядеть так, как показано в листинге 2.1, являющемся файлом `MyWidget.h`.

Листинг 2.1. Пример интерфейса (MyWidget.h)

```
#import <Foundation/Foundation.h>
@interface MyWidget : BaseWidget
{
    BOOL isPoweredOn;
    @private float speed;
    @protected float mass;
    @protected float gyroscope;
}
```

```
+ (id) alloc;
+ (BOOL) needsBatteries;
- (BOOL) powerOn;
- (void) setSpeed: (float) _speed;
- (void) setSpeed: (float) _speed withMass: (float) _mass;
- (void) setSpeed: (float) _speed withGyroscope: (float) _gyroscope;
@end
```

Все важные семантические элементы в этом файле будут объяснены в следующих разделах.

Импорт

Директива препроцессора `#import` замещает обычную директиву `#include` (хотя `#include` все равно может использоваться). Единственное преимущество использования `#import` состоит в том, что она является встроенной логикой, гарантирующей, что один и тот же ресурс никогда не будет включен более одного раза. Такой подход замещает обходной способ использования макро-флагов, обычно применяемый в коде C:

```
#ifndef _MYWIDGET_H
#define _MYWIDGET_H
...
#endif
```

Объявление интерфейсов

Интерфейс объявляется оператором `@interface` с последующим указанием имени интерфейса и базового класса (если таковой есть), от которого он был порожден. Блок заканчивается оператором `@end`.

Методы

Методы объявляются вне структур в фигурных скобках. Знак плюса (+) определяет метод как статический, а знак минуса (−) объявляет экземпляр метода. Таким образом, метод `alloc` (для назначения нового объекта) будет вызываться с помощью ссылки напрямую в класс `MyWidget`, в то время как методы, являющиеся специфическими для экземпляра класса `MyWidget`, на-

пример, `needBatteries` и `powerOn`, будут вызваны на экземпляр, возвращаемый `alloc`.

Каждый объявленный аргумент для метода представлен типом данных, локальными именами переменных и необязательно внешними именами переменных. Примерами внешних имен переменных в листинге 2.1 являются `withMass` и `WithGyroscope`. Вызываемая функция (`notifier`), которая вызывает метод, ссылается на внешние имена переменных, но внутри метода на аргументы ссылаются, используя их локальные имена переменных. Таким образом, метод `setSpeed` использует переменную `local_mass` для получения значения, передаваемого как `withMass`.

Если в объявлении нет внешнего имени переменной, то на переменную ссылаются только с двоеточием, например, `:10.0`.

Реализация

Суффиксом кода для исходных файлов Objective-C является `.m`. Реализация скелета простейшего класса из последнего раздела может быть такой, как показано в листинге 2.2 (файл `MyWidget.m`).

Листинг 2.2. Пример реализации (`MyWidget.m`)

```
#import "MyWidget.h"

@implementation MyWidget

+ (id)alloc {
}

+ (BOOL)needsBatteries {
    return YES;
}

- (BOOL)powerOn {
    isPoweredOn = YES;
    return YES;
}
```

```
- (void)setSpeed:(float)_speed {
    speed = _speed;
}

- (void)setSpeed:(float)_speed withMass:(float)_mass {
    speed = _speed;
    mass = _mass;
}

- (void)setSpeed:(float)_speed withGyroscope:(float)_gyroscope {
    speed = _speed;
    gyroscope = _gyroscope;
}

@end
```

Так как интерфейс заключен в собственный блок кода, то реализация начинается с оператора `@implementation` и заканчивается оператором `@end`. Эта весьма распространенная практика в C++ использовать для переменных экземпляра в качестве префикса `m_`, чтобы публичные методы могли принимать имена переменных. Это облегчает повторное использование чужого кода, поскольку становится возможным определить назначение переменной по ее имени. Поскольку Objective-C разрешает использование внешних имен переменных, то метод может предоставлять разработчику для использования осмысленное имя, в то время как внутри использовать собственное имя. Это истинное имя затем может быть использовано внутри объекта, а локальное имя переменной метода имеет в качестве префикса символ подчеркивания, например, `_speed`.

Категории

Objective-C привносит в объектно-ориентированное программирование новый элемент под названием *категории* (categories). Категории были разработаны для того, чтобы решить проблему, при которой базовые классы рассматриваются как хрупкие, чтобы не допустить разрушение более сложных производных классов внешне безобидными изменениями. Когда программа достигает определенного размера, разработчик зачастую боится трогать

меньшие базовые классы, поскольку без внимательного изучения всего приложения достаточно трудно определить, какие изменения безопасны. Категории предоставляют механизм для добавления функциональности меньшим классам, не затрагивая другие объекты.

Класс категории может помещаться "поверх" меньшего класса, добавляя или замещая методы внутри базового класса. Это можно сделать без перекомпиляции или даже при отсутствии доступа к исходному коду базовых классов. Категории позволяют базовым классам быть расширенными внутри ограниченного пространства, чтобы любые объекты, использующие базовый класс (а не категорию), продолжали видеть оригинальную версию. С точки зрения разработки это существенно облегчает процесс усовершенствования класса, написанного другим разработчиком. Во время исполнения части кода, использующие категорию, будут видеть новую версию класса, а код, использующий базовый класс напрямую, будет видеть только оригинальную версию.

Различие между категориями и наследованием такое же, как между тюнингом вашей машины и украшением ее для использования в качестве праздничной платформы на параде. Когда вы модернизируете вашу спортивную машину, то добавляете внутрь автомобиля новые компоненты, в результате чего он начинает иначе функционировать. Иногда какие-либо компоненты даже полностью вытаскиваются и заменяются на новые. Факт добавления нового компонента в двигатель, например, турбо, влияет на функционирование всего автомобиля. Это принцип работы наследования.

С другой стороны, категории больше походят на праздничную платформу, в которой автомобиль остается абсолютно нетронутым, а макеты из картона, папье-маше прикрепляются к автомобилю с внешней стороны, делая его неузнаваемым. В контексте парада автомобиль выступает в роли совершенно иного "животного", но для механика он все тот же ваш старый автомобиль, которым вы управляете много лет.

Итак, категории добавляют некоторую свободу, но в то же время призваны устранить ситуации, при которых вносимые в базовый класс изменения разрушают существующее приложение. Путем создания категории приложения, использующие базовый класс `MyWidget`, продолжают видеть оригинальный класс, в то время как новые приложения вместо него будут использовать категорию. Приведенный далее пример создает категорию `MySpaceWidget` поверх существующего базового класса `MyWidget`. Поскольку нам потребуется возможность уничтожения различных вещей, то мы добавили метод

selfDestruct. Кроме того, данная категория замещает метод powerOn собственным. Сравните применение здесь круглых скобок для MySpaceWidget, содержащего класс, с использованием двоеточия в листинге 2.1 для обеспечения наследования:

```
#import "MyWidget.h"

@interface MyWidget (MySpaceWidget)
- (void)selfDestruct;
- (BOOL)powerOn;
@end
```

В листинге 2.3 приведен полный исходный файл, реализующий категорию.

Листинг 2.3. Пример категории (MySpaceWidget.m)

```
#import "MySpaceWidget.h"

@implementation MyWidget (MySpaceWidget)

- (void)selfDestruct {
    isPoweredOn = 0;
    speed = 1000.0;
    mass = 0;
}

- (BOOL)powerOn {
    if (speed == 0) {
        isPoweredOn = YES;
        return YES;
    }

    /* Не включать питание, если космический корабль движется. */
    return NO;
}

@end
```

Маскировка

В Objective-C класс подкласса может *маскироваться* (pose) под один из его суперклассов, фактически замещая его как получателя всех сообщений. Это похоже на подмену (overriding) с единственным отличием, состоящим в том, что подменяется весь класс целиком, а не единичный метод. Маскирующемуся классу нельзя объявлять никаких переменных, хотя он и может подменить или заменить существующие методы. Маскировка похожа на категории в том, что она позволяет разработчику дополнять существующий класс во время исполнения.

В предыдущих примерах были созданы классы виджетов (widgets). Теперь после создания этих виджетов в некоторый момент была открыта вечная энергия. Это позволило многим новым виджетам стать автономными, в то время как некоторые устаревшие виджеты продолжают работать на аккумуляторах. Поскольку автономные виджеты имеют существенное количество совершенно другого кода, то был порожден новый объект `MyAutonomousWidget`, чтобы подменить всю измененную функциональность, например, статический метод `needsBatteries` (см. листинги 2.4 и 2.5).

Листинг 2.4. Пример интерфейса для постановки (`MyAutonomousWidget.h`)

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

@interface MyAutonomousWidget : MyWidget
{
}

+ (BOOL)needsBatteries;
@end
```

Листинг 2.5. Пример реализации для постановки (`MyAutonomousWidget.m`)

```
#import "MyAutonomousWidget.h"

@implementation MyAutonomousWidget
```

```
+ (BOOL)needsBatteries {  
    return NO;  
}  
@end
```

Вместо того чтобы изменять весь существующий код для использования этого класса, автономный класс может просто маскироваться под класс виджета. Метод `class_poseAs` вызывается из основной программы или другого метода высокого уровня, чтобы вызвать следующее поведение:

```
MyAutonomousWidget *myAutoWidget = [ MyAutonomousWidget alloc ];  
MyWidget *myWidget = [ MyWidget alloc ];  
class_poseAs(myAutoWidget, myWidget);
```

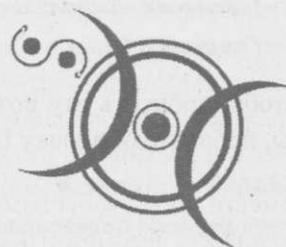
С этого момента все остальные методы, замененные нами в маскирующемся классе (чтобы изменить способ, которым мы общаемся с автономными устройствами), будут выступать в роли оригинальных базовых классов.

Для дальнейшего изучения

Чтобы узнать больше о программировании на Objective-C, обратитесь к приведенным далее великолепным ресурсам от O'Reilly:

- ❑ James Duncan Davidson, "Learning Cocoa with Objective-C", Second Edition
<http://www.oreilly.com/catalog/learncocoa2/>;
- ❑ Andrew M. Duncan, "Objective-C Pocket Reference"
<http://www.oreilly.com/catalog/objectcpr/>.

ГЛАВА 3



Введение в UIKit

UIKit — самая большая оболочка iPhone с точки зрения размера файлов, и это вполне справедливо, поскольку она отвечает за все функции пользовательского интерфейса от создания окон и текстовых полей до распознавания множественных касаний и аппаратных сенсоров. Все графические удобства, делающие iPhone простым в использовании, основаны на оболочке UIKit, что придает интерфейсу iPhone изысканность и единообразие. Для всех приложений iPhone доступны одни и те же UIKit API, поэтому понимание того, как пользоваться этой оболочкой, позволит вам применять все те инструменты, с помощью которых собственные приложения Apple становятся такими эффектными.

UIKit — это больше, чем просто набор инструментов пользовательского интерфейса; это также основание для исполнения GUI-приложений iPhone. Когда приложение запущено, его функция `main()` обрабатывает объект `UIApplication` в рамках UIKit. Этот класс — базовый для всех приложений, имеющих в iPhone пользовательский интерфейс, и он предоставляет доступ уровня приложений к функциям более высокого уровня iPhone. Помимо этого, общие сервисы уровня приложения, например, приостановка, возобновление и завершение, являются функциями объекта `UIApplication`.

Чтобы начать использование UIKit, ваше приложение должно подключиться к оболочке UIKit. Как оболочка, UIKit является неким вариантом библиотеки совместного использования. Поэтому, используя пакет инструментов компилятора, UIKit может быть подключен к вашему приложению путем добавления к аргументам компилятора, описанным в *главе 2*, следующих аргументов:

```
$ arm-apple-darwin9-gcc -o MyApp MyApp.m -lobjc -framework CoreFoundation
```

```
-framework Foundation  
-framework UIKit
```

Чтобы добавить эту возможность в пример make-файла из предыдущей главы, добавьте оболочку UIKit в раздел флагов компоновщика:

```
LDFLAGS = -lobjc  
-framework CoreFoundation  
-framework Foundation  
-framework UIKit
```

Основные элементы пользовательского интерфейса

Целью этой главы является подготовка вас к работе с основными составляющими пользовательского интерфейса. UIKit поддерживает приведенные далее элементы пользовательского интерфейса. Более сложные функциональные возможности UIKit описаны в *главе 7*.

- **Окна и виды** являются базовыми классами для создания пользовательского интерфейса любого типа. Окно (window) представляет собой геометрическое пространство на экране, а вид (view) выступает в роли контейнера для других объектов. Все небольшие компоненты пользовательского интерфейса, например, панели навигации, кнопки и текстовые поля, привязаны к виду, а уже этот вид прикреплен к какому-либо окну. Окно можно рассматривать как рамку картины, а вид — как само полотно. Окно может иметь только один вид, но виды могут содержать более мелкие подвиды, включая другие виды.

Управляющий вид (controlling view) — это вид, который управляет тем, как другие виды отображаются на экране. Управляющий вид выполняет переходы к другим видам и реагирует на события, происходящие на экране.

- **Текстовые виды** (text views) — это специальные классы видов, представляющие окно редактора, для просмотра или редактирования текста или HTML. Очень хорошим примером простейшего текстового вида является приложение Блокнот (Notepad). Вообще говоря, текстовые виды прикидываются скромными и в своем репертуаре редко используют более эффектные инструменты пользовательского интерфейса.

- **Панели навигации.** Пользовательский интерфейс iPhone рассматривает различные экраны, как если бы они были страницами в книге. Панели навигации (navigation bars) зачастую используются для предоставления пользователю визуальных напоминаний, чтобы позволить пользователю вернуться назад к предыдущему виду, обеспечить его кнопками для изменения элементов на текущей странице экрана и предоставить отформатированные заголовки к просматриваемой странице экрана. Панели навигации можно обнаружить практически в каждом изначально загруженном приложении iPhone.
- **Переходы.** Для функционирования любого приложения, как правило, недостаточно всего одной страницы, особенно на мобильном устройстве. Согласно принципам дружелюбности пользовательских интерфейсов Apple в iPhone были введены переходы окон (window transitions), чтобы создать у пользователя ощущение перемещения по их приложениям словно по страницам книги. Переходы окон используются для создания этого визуального перехода от одного окна к другому, и они предоставляют целый ряд различных переходов от знакомого всем эффекта переворачивания страницы до постепенного исчезновения или скручивания.
- **Листы действий (action sheets)** являются аналогом всплывающих окон напоминаний в iPhone. Листы действий появляются как модальные окна, выскальзывающие снизу в тех случаях, когда какая-либо операция требует привлечения внимания пользователя. Чаще всего их можно увидеть на предустановленных на iPhone приложениях при попытке пользователя удалить несколько элементов или важные данные, например, голосовую почту. Листы действий могут быть запрограммированы задавать пользователю любые вопросы и предоставлять ему любое количество различных вариантов действий. Они весьма полезны в тех частях приложений, которые требуют немедленной реакции.
- **Таблицы (tables)** на самом деле являются списками, которые можно использовать для отображения файлов, сообщений или других типов коллекций. Они применяются для выделения одного или нескольких элементов методом, используемым в списках. Объекты таблицы очень гибки и позволяют разработчику определять внешний вид и поведение ячейки таблицы.
- **Строка состояния (status bar)** — это небольшая панель, расположенная наверху экрана iPhone и отображающая время, уровень зарядки батареи и качество сигнала. Цвет, прозрачность и положение строки состояния можно менять и настраивать; кроме того, можно сделать так, чтобы стро-

ка состояния отображала значки, уведомляющие пользователя о том или ином состоянии приложения.

- **Бейджи приложений.** Те приложения, которым необходимо в определенные промежутки времени уведомлять пользователя о каких-либо событиях, имеют возможность отображать на основном экране приложений бейджи (badges). Это позволяет проинформировать пользователя о том, что какое-либо приложение требует внимания, или же о том, что для пользователя имеется сообщение или какая-либо другая информация. Эта возможность очень активно применяется приложениями, использующими для доставки сообщений сеть EDGE.
- **Значки строки состояния.** В зависимости от потребностей приложения можно менять стиль, прозрачность и положение строки состояния. Также на строку состояния могут быть добавлены изображения, информирующие пользователя об имеющихся местах операций (например, об аварийных или фоновых процессах).

Окна и виды

Основным компонентом пользовательского интерфейса является окно. Окно (window) — это область отображения, рамка изображения. `UIWindow` — это базовый класс iPhone, порожденный от функций более низкого уровня, отвечающих на события мыши, жесты (gestures) и другие типы событий, относящиеся к этому окну. Класс `UIWindow` в конечном счете ответственен за хранение содержимого объекта `UIView` (изображение, которое размещается в рамке окна). `UIView` является базовым классом, от которого порождены многие другие типы классов экрана. Само по себе окно может только хранить один объект, в то время как объект `UIView` создан для того, чтобы содержать в себе множество различных типов подобъектов, включая другие виды. Эти два класса идут рука об руку, и оба они должны отображать что-либо на экране iPhone.

Создание окна и вида

Прежде чем вы сможете что-либо отобразить на экране iPhone, вы должны создать окно для хранения содержимого, а чтобы построить окно, вам необходима область отображения. Область отображения (display region) — термин, обозначающий прямоугольник и представляющий часть экрана, в кото-

рой должно отображаться окно. Сама же основная структура является структурой прямоугольника `CGRect`, которая содержит две порции информации: координаты левого верхнего угла окна и размер окна (его ширину и высоту). Каждый объект, отображаемый на экране, имеет область отображения, задающую его часть экрана. Чаще всего области отображения задаются во время инициализации объекта посредством метода `initWithFrame`. Иначе они должны задаваться после использования вездесущего метода `setFrame`. В случае основного окна область отсчитывается от самого экрана, однако все последующие объекты (включая вид окна) отсчитываются от объекта, их содержащего. Поскольку все другие объекты являются вложенными в вид, то области их объектов отсчитываются от вида, и т. д.

Приложение использует при отображении весь экран iPhone, поэтому окну должен соответствовать набор координат, отражающих вид области всего экрана. Эта область может быть получена из статического метода, находящегося внутри класса `UIHardware`:

```
CGRect windowRect = [ UIHardware fullScreenApplicationContentRect ];
```

Эта область (`windowRect`) затем используется для создания и инициализации нового объекта `UIWindow`:

```
UIWindow *window = [ UIWindow alloc ];  
[ window initWithContentRect: windowRect ];
```

Итак, теперь создана рамка окна, но она ничего не содержит. Необходим объект, который сможет визуализировать внутри нее содержимое. Чтобы заполнить окно, должен быть создан объект `UIView`. Поскольку окно может только хранить один объект вида, то область отображения для него должна точно так же использовать весь экран целиком, чтобы он заполнял все окно:

```
CGRect viewRect = [ UIHardware fullScreenApplicationContentRect ];  
viewRect.origin.x = viewRect.origin.y = 0.0;
```

```
UIView *mainView = [ [ MainView alloc ] initWithFrame: viewRect ];
```

Отображение вида

Теперь создана пара — окно и вид, но вид еще не отображен на экране. Для этого определим окно и прикажем окну отобразиться:

```
[ window setContentView: mainView ];  
[ window orderFront: self ];
```

```
[ window makeKey: self ];  
[ window _setHidden: NO ];
```

Самое бесполезное приложение

Прежде чем мы перейдем к "Hello, World!", нам потребуется еще более бесполезное приложение: "Hello, Window!". Это приложение не делает ничего, кроме как просто создает пару окна и вида. На самом деле, поскольку базовый класс `UIView` является всего лишь контейнерным классом, то он даже текст не может отобразить. Все, что вы увидите, — это черный экран. Единственное, чем это приложение полезно, — это несколько первых строк кода, которые будет использовать любое приложение GUI на iPhone.

Это приложение, показанное в листингах 3.1 и 3.2, может быть скомпилировано из пакета инструментов с помощью следующей командной строки:

```
$ arm-apple-darwin9-gcc -o MyExample MyExample.m -lobjc   
-framework CoreFoundation -framework Foundation -framework UIKit
```

Листинг 3.1. Пример окна и вида (MyExample.h)

```
#import <CoreFoundation/CoreFoundation.h>  
#import <UIKit/UIKit.h>  
  
@interface MyApp : UIApplication  
{  
    UIWindow *window;  
    UIView *mainView;  
}  
- (void)applicationDidFinishLaunching:  
    (NSNotification *)aNotification;  
@end
```

Листинг 3.2. Пример окна и вида (MyExample.m)

```
#import "MyExample.h"  
  
int main(int argc, char **argv)
```

```
{
    NSAutoreleasePool *autoreleasePool = [
        [ NSAutoreleasePool alloc ] init
    ];
    int returnCode = UIApplicationMain(argc, argv, @"MyApp", @"MyApp");
    [ autoreleasePool release ];
    return returnCode;
}

@implementation MyApp

- (void)applicationDidFinishLaunching:(NSNotification *)aNotification {
    UIWindow *window = [ [ UIWindow alloc ] initWithContentRect:
        [ UIHardware fullScreenApplicationContentRect ]
    ];

    CGRect windowRect =
        [ UIHardware fullScreenApplicationContentRect ];
    windowRect.origin.x = windowRect.origin.y = 0.0f;

    UIView *mainView = [ [ UIView alloc ] initWithFrame: windowRect ];

    [ window setContentView: mainView ];
    [ window orderFront: self ];
    [ window makeKey: self ];
    [ window setHidden: NO ];
}

@end
```

Как это работает

Приложение "Hello, Window!" работает следующим образом:

1. При запуске приложения вызывается его функция `main()`, точно так же, как это происходит в обычной C-программе. Происходит переход в страну Objective-C и порождение класса `MyApp`, являющегося производным от класса `UIApplication` UIKit. Функция `main()` также отвечает за инициализацию ав-

томатически освобождаемого пула (auto-release pool). Автоматически освобождаемые пулы широко применяются в платформе Сосоа для освобождения объектов, которые при своем создании были построены как autorelease. Тем самым приложению указывается просто уничтожить их после того, как упадет надобность в них, и освободить занимаемые ими ресурсы.

2. Метод `applicationDidFinishLaunching` класса `UIApplication` вызывается основной платформой объекта приложения после инициализации объекта. Именно в этом месте зарождается жизнь приложения Objective-C.
3. Метод `fullScreenApplicationContentRect` класса `UIHardware` вызывается для того, чтобы вернуть координаты и размер физического экрана. Затем это используется для создания нового окна, в котором хранится основной вид приложения.
4. Далее создается основной вид с использованием области отображения, начинающейся в точке (0, 0) — левый верхний угол окна. Вид становится содержимым окна.
5. Затем окну дается команда перейти на передний план и отобразиться. Таким образом, отображается вид, который на данный момент не имеет содержимого.

Порождение от `UIView`

Пример "Hello, World!" показал нам, как мало кода необходимо для создания и отображения пары "окно/вид". Поскольку сам по себе класс `UIView` является всего лишь базовым классом, то он ничего не отображает. Чтобы создать полезное приложение, можно породить от `UIView` новый класс, подменив его методы для добавления функциональности. Этот управляющий вид может затем отображать другие объекты, например, текстовые поля, изображения и т. д.

Чтобы породить подкласс от `UIView`, напишите новые интерфейс и реализацию, объявляющие подкласс. Приведенный далее фрагмент кода создает подкласс `MainView`:

```
@interface MainView : UIView
{
}
- (id)initWithFrame:(CGRect) rect;
- (void)dealloc;
@end
```

Должны быть подменены, по меньшей мере, два метода класса `UIView`. Метод `initWithFrame` вызывается, когда вид порождается впервые, и используется для инициализации класса вида. В метод передается область отображения для определения координат вида (относительно его родителя) и размера, в котором он должен отображаться. Любой код, инициализирующий переменные или другие объекты, должен останавливаться на этом методе. Второй метод, `dealloc`, вызывается при уничтожении объекта `UIView`. Все ресурсы, занятые ранее в рамках вашего класса, должны быть здесь освобождены.

Эти два метода являются основой для всех других действий в пределах класса вида. Вот шаблоны для них:

```
@implementation MainView
- (id)initWithFrame:(CGRect)rect {

    self = [super initWithFrame: rect];
    if (nil != self) {
        /* Здесь инициализируются переменные экземпляра */
        /* Здесь выделяются первоначальные ресурсы */
    }

    return self;
}

- (void)dealloc
{
    /* Здесь освобождаются все ресурсы */
    [self dealloc];
    [super dealloc];
}
@end
```

Второе самое бесполезное приложение

Теперь, когда вы знаете, как породить класс `UIView`, у вас есть все, что вам нужно для написания приложения, которое что-либо делает — пусть даже и нечто самое бесполезное. Согласно традиции мы сейчас представляем формально бесполезное приложение "Hello, World!"

Это приложение, показанное в листингах 3.3 и 3.4, может быть построено с использованием тех же самых аргументов командной строки, что и в предыдущем примере:

```
$ arm-apple-darwin9-gcc -o MyExample MyExample.m -lobjc  
-framework CoreFoundation -framework Foundation -framework UIKit
```

Листинг 3.3. Пример "Hello, World!" (MyExample.h)

```
#import <CoreFoundation/CoreFoundation.h>  
#import <UIKit/UIKit.h>  
#import <UIKit/UITextView.h>  
  
@interface MainView : UIView  
{  
    UITextView *textView;  
}  
- (id)initWithFrame:(CGRect)frame;  
- (void)dealloc;  
@end  
  
@interface MyApp : UIApplication  
{  
    UIWindow *window;  
    MainView *mainView;  
}  
- (void)applicationDidFinishLaunching:  
    (NSNotification *)aNotification;  
@end
```

Листинг 3.4. Пример "Hello, World!" (MyExample.m)

```
#import "MyExample.h"  
  
int main(int argc, char **argv)  
{  
    NSAutoreleasePool *autoreleasePool = [  
        [NSAutoreleasePool alloc] init
```

```
];
int returnCode = UIApplicationMain(argc, argv, @"MyApp", @"MyApp");
[ autoreleasePool release ];
return returnCode;
}

@implementation MyApp

- (void)applicationDidFinishLaunching:(NSNotification *)aNotification {
    window = [ [ UIWindow alloc ] initWithContentRect:
        [ UIScreen fullScreenApplicationContentRect ]
    ];

    CGRect rect = [ UIScreen fullScreenApplicationContentRect ];
    rect.origin.x = rect.origin.y = 0.0f;

    mainView = [ [ MainView alloc ] initWithFrame: rect ];

    [ window setContentView: mainView ];
    [ window orderFront: self ];
    [ window makeKey: self ];
    [ window _setHidden: NO ];
}

@end

@implementation MainView

- (id)initWithFrame:(CGRect)rect {

    self = [ super initWithFrame: rect ];
    if (nil != self) {

        textView = [ [ UITextView alloc ] initWithFrame: rect ];
        [ textView setText: @"Hello, World!" ];
        [ self addSubview: textView ];
    }
}
```

```
    return self;
}

- (void)dealloc
{
    [ self dealloc ];
    [ super dealloc ];
}

@end
```

Как это работает

Пример "Hello, World!" содержит все, что вы видели до этого момента, а также новый вид, созданный для отображения текста:

1. Приложение порождается точно так же, как и раньше, — путем вызова программной функции `main()`, которая создает экземпляр `MyApp`.
2. Вместо создания родового класса `UIView` приложение порождает собственный класс `MainView`, являющийся производным от `UIView`.
3. Вызывается метод `initWithFrame` класса `MainView`, который в свою очередь вызывает свой родительский класс (`UIView`) и собственный метод `initWithFrame`, чтобы позволить `UIView` сделать работу по созданию самого вида.
4. `UITextView`, о котором вы узнаете более подробно в *разд. "Текстовые виды" далее в этой главе*, создается и присоединяется к объекту `MainView`. Этот текстовый вид предоставляет текст "Hello, World!"
5. Окну дается указание отобразить объект `MainView` и присоединенный к нему объект `UITextView`.

Для дальнейшего изучения

Теперь, когда в вашем приложении есть то, на что можно посмотреть, сделайте следующее.

- Попробуйте изменить начало координат и размер рамки, передаваемые в `mainView`. Что произойдет с окном и его потомком? А как насчет того, чтобы изменить начало координат отображения `textView`?

- Проверьте наличие прототипов `UIWindow.h` и `UIView.h` в вашем каталоге `include`. Вы найдете их в папке `/toolchain/sys/usr/include/UIKit/`.

Текстовые виды

Класс `UITextView` основан на `UIView`, но его функциональность расширена для предоставления возможностей отображения и редактирования текста, прокручивания и хранения различных шрифтов и цветов. Текстовые виды можно легко испортить, поэтому рекомендуется использовать их исключительно для текстовых частей приложений, например, электронных книг, раздела программы, где нужно делать заметки, или же информационной странички для отображения неструктурированной информации.

Объект `UITextView` наследует от `UIScrollView`, являющегося родовым классом прокручивания. Это означает то, что текстовый вид сам по себе уже обладает всей функциональностью прокручивания, поэтому разработчик может сфокусироваться на отображении содержимого, а не на программировании панелей прокрутки. Класс `UIScrollView` наследует от `UIView`, который является, как было упомянуто в предыдущем разделе, базовым классом для всех классов видов.

Создание текстового вида

Поскольку `UITextView` порожден от `UIView`, то он создан по подобию объектов основного вида, созданных в предыдущем разделе, т. е. с помощью метода `initWithFrame:`:

```
UITextView *textView = [ [ UITextView alloc ]  
    initWithFrame: viewRect ];
```

После создания вида можно задать целый ряд различных свойств.

Редактирование

Если текстовый вид будет использоваться для хранения пользовательского ввода, то он должен быть редактируемым:

```
[ textView setEditable: YES ];
```

Если же вид просто отображает данные, которые не нужно редактировать, то эта возможность должна быть отключена.

Поля

Размер верхнего поля — это единственная настройка, которая может быть установлена в текстовом виде. Это значение представляет число пикселей, которые необходимо отступить от верха текстового вида для вывода текста:

```
[ textView setMarginTop: 20 ];
```

Задание содержимого

Задать содержимое текстового вида могут два различных метода: `setText` и `setContentToHTMLString`. Как ясно из названий, вызов метода `setText` приводит к тому, что содержимое отображается как текст, а `setContentToHTMLString` рассматривает содержимое как Web-страницу. Если вы попытаетесь вызвать `setText` с использованием содержимого в формате HTML, то оно отобразится как исходный код HTML.

Вот пример текстового отображения:

```
[ textView setText: @"Hello, world!" ];
```

а HTML может быть отображен так:

```
[ textView setContentToHTMLString:  
    @"<b><center>Hello, World!</center></b>" ];
```

Отображение текстового вида

Текстовые виды, как правило, присоединены к основному виду окна. Это позволяет позднее добавлять в тот же самый родительский вид другие подвиды, например, панели навигации и элементы управления:

```
[ mainView addSubview: textView ];
```

Пример:

отображение отказа от ответственности iPhone

Каждый iPhone, поставляемый Apple, имеет достаточно длинный файл HTML, содержащий официальный отказ от ответственности Apple. Он отображается в правовом разделе приложения **Settings**. Приводимый далее пример берет этот файл и отображает его в текстовом поле.

Чтобы скомпилировать это приложение, вам потребуется включить оболочку Core Graphics, содержащую процедуры, необходимые для смешивания цветов.

Данное приложение, показанное в листингах 3.5 и 3.6, может быть построено с помощью следующей командной строки пакета инструментов:

```
$ arm-apple-darwin9-gcc -o MyExample MyExample.m -lobjc \
    -framework CoreFoundation -framework Foundation -framework UIKit \
    -framework CoreGraphics
```

Листинг 3.5. Пример UITextView (MyExample.h)

```
#import <CoreFoundation/CoreFoundation.h>
#import <Foundation/Foundation.h>
#import <UIKit/UIKit.h>
#import <UIKit/UITextView.h>

@interface MainView : UIView
{
    UITextView *textView;
}

- (id)initWithFrame:(CGRect)frame;
- (void)dealloc;

@end

@interface MyApp : UIApplication
{
    UIWindow *window;
    MainView *mainView;
}

- (void)applicationDidFinishLaunching:(NSNotification *)aNotification;
@end
```

Листинг 3.6. Пример UITextView (MyExample.m)

```
#include <stdio.h>
#import "MyExample.h"
```

```
int main(int argc, char **argv)
{
    NSAutoreleasePool *autoreleasePool = [
        [NSAutoreleasePool alloc] init
    ];
    int returnCode = UIApplicationMain(argc, argv, @"MyApp", @"MyApp");
    [autoreleasePool release];
    return returnCode;
}
```

```
@implementation MyApp
```

```
- (void)applicationDidFinishLaunching:(NSNotification *)aNotification {
    UIWindow *window = [ [UIWindow alloc] initWithContentRect:
        [UIHardwareFullScreenApplicationContentRect]
    ];
```

```
    CGRect rect = [UIHardwareFullScreenApplicationContentRect];
    rect.origin.x = rect.origin.y = 0.0f;
```

```
    UIView *mainView = [ [MainView alloc] initWithFrame: rect ];
```

```
    [window setContentView: mainView];
```

```
    [window orderFront: self];
```

```
    [window makeKey: self];
```

```
    [window _setHidden: NO];
```

```
}
```

```
@end
```

```
@implementation MainView
```

```
- (id)initWithFrame:(CGRect)rect {
    self = [super initWithFrame: rect];
    if (nil != self) {
```

```
        FILE *file;
```

```
        char buffer[262144], buf[1024];
```

```
textView = [ [ UITextView alloc ] initWithFrame: rect ];

file = fopen("/Applications/Preferences.app/English.lproj/
acknowledgements-iphone.html", "r");

if (!file) {
    CGColorSpaceRef colorSpace =
        CGColorSpaceCreateWithName(kCGColorSpaceGenericRGB);
    float opaqueRed[4] = { 1.0, 0.0, 0.0, 1 };
    CGColorRef red = CGColorCreate(colorSpace, opaqueRed);
    [ textView setTextColor: red ];

    [ textView setText: @"ERROR: File not found" ];
} else {
    buffer[0] = 0;
    while((fgets(buf, sizeof(buf), file))!=NULL) {
        strcat(buffer, buf, sizeof(buffer));
    }
    fclose(file);

    [ textView setContentToHTMLString:
        [ [ NSString alloc ] initWithCString: buffer ] ];
}

[ self addSubview: textView ];

return self;
- (void)dealloc
{
    [ self dealloc ];
    [ super dealloc ];
}
@end
```

Как это работает

Пример считывает и отображает содержимое файла следующим образом:

1. При инициализации приложения создается основной вид и вызывается его метод `initWithFrame`.
2. Метод `initWithFrame` порождает новый объект `UITextView` и назначает размер текста.
3. Файл `legal-disclaimer-iphone.html` открывается с помощью метода `fopen` POSIX C.
4. Если файл не удастся найти, то текущим становится красный цвет, а текстом окна становится сообщение об ошибке.
5. Если файл найден, то он считывается в текстовый буфер, а затем устанавливается в качестве HTML содержимого окна.
6. Сам текстовый вид добавляется как подвид в управляющий вид, в котором он отображается пользователю.

Для дальнейшего изучения

Вот несколько эффективных способов протестировать этот пример.

- Скопируйте HTML-страницу вашего любимого Web-узла на iPhone с помощью SCP. Модифицируйте этот пример, чтобы он отображал ваш файл. Какие ограничения имеются у `UITextView` при отображении HTML?
- Какую еще информацию о стилях шрифта вы можете передать в метод `setFont` текстового вида? Изучите CSS-спецификацию W3C, расположенную по адресу: <http://www.w3.org>.
- Проверьте наличие прототипов `UITextView.h` в каталоге `include` пакета инструментов. Вы найдете ее в папке `/toolchain/sys/usr/include/UIKit/`.

Панели навигации

iPhone не поддерживает панели инструментов в традиционном понимании рабочего стола. Поскольку каждый экран приложения рассматривается как страница в книге, то Apple создала собственную версию панели инструмен-

тов для iPhone, чтобы она имела более книжный вид. В отличие от панелей инструментов, которые могут отображать множество различных значков, панели навигации имеют определенные ограничения и могут содержать только заголовок страницы (например, "Сохраненные сообщения"), кнопки перехода к родительским страницам и текстовые кнопки для контекстно-зависимых функций, таких как включение громкой связи или очистки списка элементов. Панели навигации также могут поддерживать элементы управления для добавления кнопок с вкладками, например, кнопки вызова **Все** или **Не принятые** при просмотре последних вызовов.

Создание панели навигации

Для создания панели навигации породите ее как объект и вызовите ее метод `initWithFrame` точно так же, как и `UIView`:

```
UINavigationController *navBar = [ [UINavigationController alloc]
initWithFrame: CGRectMake(0, 0, 320, 48)
];
```

Приведенный фрагмент создает панели навигации высотой в 48 пикселей (по умолчанию) в точке (0, 0) — левый верхний угол родительского вида. Это общепринятое соглашение, но панель навигации может быть создана в любом месте окна. Используя различные вертикальные отступы, вы можете поместить панель навигации внизу окна или под другой панелью навигации.

Чтобы узнавать о том, что на панели навигации что-либо произошло, например нажата та или иная кнопка, пользуйтесь делегатом панели навигации. Делегат (`delegate`) — это объект, который действует от имени другого объекта. Установив значение делегата панели навигации в `self`, вы можете заставить его отправлять события, например нажатия кнопок, объекту, создавшему панель навигации:

```
[ navBar setDelegate: self ];
```

Анимацией являются простые исчезающие переходы, происходящие при переходе от старого набора кнопок к новому набору, например, если кнопка после нажатия меняет свой вид. Эти плавные переходы вместо резких изменений внешнего вида использует вызов `enableAnimation`:

```
[ navBar enableAnimation ];
```

После того как панель навигации будет создана и инициализирована, ее заголовок и кнопки можно настраивать. Кроме того, их можно менять даже после отображения панели навигации; панель изменит свой внешний вид, чтобы отразить новые настройки.

Установка заголовка

Заголовок панели навигации отображается в виде крупного текста белого цвета посередине панели. Заголовок часто используется для того, чтобы проинформировать конечного пользователя о том, какого рода информация отображена в окне, например, "Сохраненные сообщения".

Заголовок создается как объект `UINavigationController`. Это базовый класс для всего, что изначально присоединено к панели навигации, включая кнопки и другие объекты. Когда заголовок добавляется к панели навигации, объект `UINavigationController` проталкивается в нее как объект в стеке. В связи с этим вам лучше хранить указатель на заголовок где-нибудь внутри программы. Это позволит вам возвращаться и менять заголовок без необходимости предварительно выталкивать его из стека, что приводило бы к перенастройке всей панели навигации:

```
UINavigationController *navItem = [ [ UINavigationController alloc ]
initWithTitle:@"My Example" ];
[ navBar pushViewController: navItem ];
```

Здесь `navItem` указывает на только что созданный объект `UINavigationController` и имеет значение заголовка "My Example". Когда пользователь делает что-либо, что приводит к изменению заголовка, `navItem` может быть изменено на что-либо другое:

```
[ navItem setTitle: @"Another Example" ]
```

Кнопки и стили кнопок

Кнопки могут быть добавлены на панель навигации справа и слева. Поскольку пространство панели ограничено, то единственные кнопки, которые должны быть добавлены, — это те, которые предназначены для функций, специфических для отображаемой страницы. Кнопки панели навигации могут быть изменены в любой момент. При разрешенной анимации пользователь увидит плавные анимированные переходы от старого к новому набору кнопок.

Используя пример объекта `navBar`, созданный ранее, добавим на навигационную панель две кнопки — **Good** и **Evil**:

```
[ navBar showLeftButton:@"Good" withStyle: 0
      rightButton:@"Evil" withStyle: 0 ];
```

Если вам нужна только одна кнопка, то вы можете заменить строку второй кнопки на значение `nil`. Например, если вы хотите предоставить пользователю на выбор только **Evil**:

```
[ navBar showLeftButton:nil withStyle: 0
      rightButton:@"Evil" withStyle: 0 ];
```

Помимо задания названий кнопок, UIKit позволяет также применять различные стили. Например, обратите внимание, что когда вы нажимаете кнопку **Speaker** в приложении телефона iPhone, то кнопка становится синей. Некоторые кнопки могут иметь форму стрелки, передавая пользователю идею о возможности возврата назад. Путем изменения значения параметра `withStyle` в вызове метода, показанного ранее, вы можете выбирать из четырех различных стилей кнопок (табл. 3.1). Обратите особое внимание на номера стилей, которые начинаются с нуля.

Таблица 3.1

Стиль	Описание
0	Стиль, заданный по умолчанию, простая серая кнопка
1	Создает кнопку в форме левосторонней стрелки (стрелки назад)
2	Кнопки синего цвета, полезен для визуального выделения или таких переключателей, как Speaker
3	Кнопки красного цвета, полезен для переключателей предупреждений, например, Delete

В будущих выпусках программного обеспечения iPhone Apple может добавлять дополнительные стили, но на сегодняшний день любые другие значения приводят к стилю по умолчанию.

Если вы вообще не планируете использовать стили, то вот быстрый способ создания кнопок панели навигации, избавляющий вас от части работы:

```
[ navBar showButtonsWithLeftTitle:@"Back"
      rightTitle: nil
      leftBack: YES ];
```

Этот способ можно использовать для создания быстрого набора кнопок без каких-либо своих стилей, за исключением кнопки со стрелкой назад, она задается путем передачи значения YES параметру `leftBack`.

Стиль панели навигации

Сама панель навигации может быть отображена одним из нескольких различных стилей. Заданный по умолчанию стиль — это стандартное серое отображение. На сегодняшний день поддерживаются три различных стиля (табл. 3.2).

Таблица 3.2

Стиль	Описание
0	Стиль, заданный по умолчанию, белый текст на сером фоне
1	Белый текст на сплошном черном фоне
2	Белый текст на прозрачном черном фоне

Стиль устанавливается с помощью вызова `setBarStyle`:

```
[ navBar setBarStyle: 0. ];
```

Отображение панели навигации

Как только вы определитесь с начальным внешним видом вашей панели навигации, можно отобразить ее в вашем приложении. Панели навигации присоединены к объекту вида, например, основного вида, созданного вами в предыдущем примере:

```
[ self addSubview: navBar ];
```

Если вы захотите скрыть панель навигации, просто уберите ее с вида:

```
[ navBar removeFromSuperview ];
```

Когда вы полностью завершите все действия с панелью навигации, ее можно убрать. Это можно сделать в методе `dealloc` класса вида:

```
[ navBar release ];
```

Перехват нажатий кнопок

Теперь ваша панель навигации отображена на экране, но ее кнопки ничего не делают. Раньше значение делегата панели навигации было установлено в `self`. Поскольку вызываемый вид реагирует на события панели навигации, то ему потребуется возможность перехватывать нажатия кнопок.

Для этого необходимо подменить метод `buttonClicked`. Как будет показано в следующих главах, метод `buttonClicked` имеется у многих различных типов объектов, но все они принимают разные аргументы. Objective-C поддерживает *полиморфизм*, что позволяет разработчику поддерживать несколько методов, имеющих одно и то же название, но различные типы аргументов. Во время исполнения для вызова будет выбираться наиболее подходящая версия метода.

Поскольку класс вида уже был установлен в качестве делегата для панели навигации, то он будет получать все события нажатий кнопок, происходящие на ней. Это потребует от класса вида наличия метода `buttonClicked`, принимающего те же самые аргументы, как если бы это была сама панель навигации:

```
- (void)navigationBar:(UINavigationController *)navbar  
    buttonClicked:(int)button;
```

Метод `buttonClicked` в классе вида уведомляется о том, принадлежит ли он панели навигации. Он вызывается с теми же самыми аргументами, с какими вызывался бы метод `buttonClicked` панели навигации, указателем на панель навигации и целым числом, определяющим номер нажатой кнопки. Значениями `button`, обозначающими левую и правую кнопки навигации, являются 1 и 0 соответственно:

```
- (void)navigationBar:(UINavigationController *)navbar buttonClicked:(int)button  
{  
    switch(button) {  
        case 1:  
            /* Здесь обрабатывается левая кнопка */  
            break;  
        case 0:  
            /* Здесь обрабатывается правая кнопка */  
            break;  
    }  
}
```

Запрещение кнопок

Если вы вдруг захотите запретить какую-либо имеющуюся кнопку панели навигации, то у вас есть для этого два способа. Вы можете еще раз вызвать один из методов отображения кнопок (`showButtonsWithLeftTitle` или `showLeftButton`), используя значение `nil`, чтобы кнопка исчезла. Либо же, если вы хотите, чтобы кнопка оставалась видимой, но недоступной, т. е. пользователь не мог ее нажать, воспользуйтесь методом `setButton:`

```
[ navbar setButton: 0 enabled: NO ];
```

Так же, как и в методе `buttonClicked`, используемый здесь номер кнопки соответствует либо левой (1), либо правой (0) кнопке.

Добавление сегментного элемента управления

Элементы управления (controls) — это небольшие независимые компоненты интерфейса, которые могут быть использованы классами UIKit. Они могут прикрепляться к различным типам объектов, позволяя тем самым разработчику добавлять окну дополнительные способы взаимодействия с пользователем. Одним общим элементом управления, который присутствует на панелях навигации поставляемых Apple приложений, является *сегментный элемент управления* (segmented control).

Во многих поставляемых приложениях вы заметите, что Apple добавила кнопки для дальнейшего разделения отображения информации. Например, панель навигации в приложении iTunes WiFi Store вывела кнопки **New Releases**, **What's Hot** и **Genres** наверх. Таким образом, осуществляется дальнейшее разделение музыкальных предпочтений пользователя. Сегментные элементы управления весьма полезны во всех аналогичных этой ситуации, в которых чрезмерное количество данных лучше упорядочить с помощью отдельных вкладок.

Далее приведен пример настройки такого элемента управления с двумя сегментами:

```
UISegmentedControl *segCtl = [ [ UISegmentedControl alloc ]  
    initWithFrame:CGRectMake(70.0, 8.0, 180.0, 30.0)  
    withStyle: 2  
    withItems: NULL ];  
[ segCtl insertSegment:0 withTitle:@"All" animated: TRUE ];
```

```
[ segCtl insertSegment:1 withTitle:@"Missed" animated: TRUE ];  
[ segCtl setDelegate:self ];
```

Затем данный элемент управления добавляется на панель навигации таким же способом, каким панель навигации добавляется к основному окну, путем добавления его как подвида. Только теперь сегментный элемент управления добавляется на панель навигации. Это означает, что смещение области отображения отсчитывается от панели навигации, а не от соответствующего вида или окна:

```
[ navBar addSubview: segCtl ];
```

Каждая кнопка в сегментном элементе управления называется *сегментом*. Доступ к выбранному сегменту можно получить с помощью вызова метода `selectedSegment` самого элемента управления:

```
int selectedSegment = [ segCtl selectedSegment ];
```

Большее количество элементов управления рассматривается в *главе 7*.

Пример: кнопка снижения громкости разговора с женой

Некто разрабатывает кнопку снижения громкости разговора своей супруги во время споров. В таком приложении панель навигации будет создаваться в каком-либо виде самого приложения с кнопкой `Mute`. При нажатии кнопки ее цвет будет меняться на красный, а также на ней будет меняться заголовок, чтобы указать то, что громкость снижена.

Листинги 3.7 и 3.8 могут быть созданы с использованием пакета инструментов с помощью следующей командной строки:

```
$ arm-apple-darwin9-gcc -o MyExample MyExample.m -lobjc  
-framework CoreFoundation -framework Foundation -framework UIKit
```

Листинг 3.7. Пример панели навигации (MyExample.h)

```
#import <CoreFoundation/CoreFoundation.h>  
#import <UIKit/UIKit.h>  
#import <UIKit/UINavigationController.h>  
#import <UIKit/UINavigationControllerItem.h>
```

```
@interface MainView : UIView
{
    UINavigationController *navBar; /* Наша панель навигации */
    UINavigationControllerItem *navItem; /* Заголовок панели навигации */
    BOOL isMuted; /* Включено ли снижение громкости? */
}

- (id)initWithFrame:(CGRect)frame;
- (void)dealloc;
- (UINavigationController *)createNavBar:(CGRect)rect;
- (void)setNavBar;
- (void)navigationBar:(UINavigationController *)navBar button-
Clicked:(int)button;
@end

@interface MyApp : UIApplication
{
    UIWindow *window;
    MainView *mainView;
}

- (void)applicationDidFinishLaunching:(NSNotification *)aNotification;
@end
```

Листинг 3.8. Пример панели навигации (MyExample.m)

```
#import "MyExample.h"

int main(int argc, char **argv)
{
    NSAutoreleasePool *autoreleasePool = [
        [NSAutoreleasePool alloc] init
    ];
    int returnCode = UIApplicationMain(argc, argv, @"MyApp", @"MyApp");
    [autoreleasePool release];
}
```

```
    return returnValue;
}

@implementation MyApp

- (void)applicationDidFinishLaunching:(NSNotification *)aNotification {
    UIWindow *window = [ [ UIWindow alloc ] initWithContentRect:
        [ UIScreen fullScreenApplicationContentRect ]
    ];

    CGRect rect = [ UIScreen fullScreenApplicationContentRect ];
    rect.origin.x = rect.origin.y = 0.0f;

    UIView *mainView = [ [ MainView alloc ] initWithFrame: rect ];

    [ window setContentView: mainView ];
    [ window orderFront: self ];
    [ window makeKey: self ];
    [ window _setHidden: NO ];
}

@end

@implementation MainView

- (id)initWithFrame:(CGRect)rect {
    self = [ super initWithFrame: rect ];
    if (nil != self) {
        isMuted = NO;

        UINavigationController *navBar = [ self createNavBar: rect ];

        /* Обновляем navBar в первый раз */
        [ self setNavBar ];

        [ self addSubview: navBar ];
    }
}
```

```
return self;
}

- (void)dealloc
{
    [ navBar release ];
    [ navItem release ];
    [ self dealloc ];
    [ super dealloc ];
}

- (UINavigationController *)createNavBar:(CGRect)rect {
    UINavigationController *newNav = [ [UINavigationController alloc]
        initWithFrame:
            CGRectMake(rect.origin.x, rect.origin.y, rect.size.width, 48.0)
    ];

    [ newNav setDelegate: self ];
    [ newNav enableAnimation ];

    /* Добавляем наш заголовок */
    UINavigationControllerItem *navItem = [ [UINavigationControllerItem alloc]
        initWithTitle:@"My Example" ];
    [ newNav pushViewController: navItem ];
    return newNav;
}

- (void)setNavBar
{
    if (isMuted == YES) {
        [ navItem setTitle: @"Spouse (Muted)" ];
        [ navBar showLeftButton:nil withStyle: 0
            rightButton:@"Mute" withStyle: 3 ];
    } else {
        [ navItem setTitle: @"Spouse" ];
    }
}
```

```
[ navbar showLeftButton:nil withStyle: 0
      rightButton:@"Mute" withStyle: 0 ];
}
}

- (void)navigationBar:(UINavigationController *)navbar buttonClicked:(int)button
{
    if (button == 0) /* Правая кнопка */
    {
        if (isMuted == YES) /* Переключатель isMuted */
            isMuted = NO;
        else
            isMuted = YES;

        [ self setNavBar ]; /* Обновляем кнопки navbar */
    }
}

@end
```

Как это работает

Этот пример демонстрирует то, как реализовать интерфейсную часть данного приложения. Вам придется написать исходный код, чтобы самостоятельно запрограммировать снижение громкости. Вот описание работы примера:

1. Посредством функции `main()` порождается приложение и возвращается экземпляр приложения, как и в предыдущем примере.
2. Создается окно с содержимым `MainView`. Оператор, создающий объект `MainView`, еще и вызывает его метод `initWithFrame`. Тем самым создается вид и панель навигации, а в качестве значения `isMuted` устанавливается `NO`. Таким образом, приложение начинает свою работу в состоянии отключенного снижения громкости.
3. Метод `initWithFrame` вызывает метод `setNavBar` для установки панели навигации в той конфигурации, которая соответствует состоянию `isMuted`. Первый вызов создает кнопки и заголовок для значения `NO`.

4. Когда пользователь коснется **Mute**, метод `buttonClicked` вызывается в виде (делегате панели). Тем самым переключается переменная `isMuted`, а затем снова вызывается `setNavBar`, что возвращает конфигурацию панели навигации в исходное состояние. В результате этого кнопка становится красной (стиль кнопки 1), а заголовок теперь содержит слово "Muted".
5. Если пользователь опять коснется кнопки **Mute**, то метод `buttonClicked` вызывается снова, возвращая `isMuted` значение `NO`. Затем панель навигации еще раз обновляется с помощью вызова `setNavBar`.

Для дальнейшего изучения

Попробуйте протестировать фрагмент кода из этого раздела и из предыдущего.

- Попробуйте изменить этот код для поддержки примера с кнопками **Good** и **Evil** так, чтобы нажатие кнопки **Good** приводило к изменению ее цвета на синий, а нажатие кнопки **Evil** приводило к изменению ее цвета на красный.
- Возьмите код `UITextView` из листинга 3.2 и добавьте две кнопки: **HTML** и **Text**. Прикосновение к каждой из кнопок должно менять текстовый вид для отображения файла в соответствующем формате.
- Проверьте наличие прототипов `UINavigationController.h` и `UINavigationControllerItem.h` в каталоге `include` пакета инструментов. Вы найдете ее в папке `/toolchain/sys/usr/include/UIKit/`. Поэкспериментируйте с различными доступными методами.

Переходные виды

Apple славится своей приверженностью соблюдать эстетичность в пользовательских интерфейсах. Эффект плавного скольжения страниц влево и вправо дает пользователю ощущение течения данных по приложению или ощущение перемещения "вперед" и "назад". Даже приложения, не имеющие структуры книжного типа, оценят возможность плавных переходов, предлагаемую `UIKit`. Переходные виды (transition views) являются объектами, позволяющими текущему виду на экране плавно смениться другим видом с очень небольшими усилиями по программированию со стороны разработчика.

Создание перехода

Переходные виды являются наследниками `UIView`, поэтому они обладают большей частью свойств обычного вида, включая рамку. Чтобы создать переход, область отображения, принадлежащая виду, передается в метод перехода `initWithFrame:`:

```
UITransitionView *transitionView = [ [ UITransitionView alloc ]  
initWithFrame: viewRect ];
```

Один и тот же переходный вид может быть использован для множества вызовов переходов и даже для множества типов переходов, поэтому, вообще говоря, для конкретной области отображения необходим только один переходный вид. Если используется панель навигации, то вы должны рассчитывать ее положение путем вычитания ее высоты и координат из области отображения переходного вида:

```
UITransitionView *transitionView = [ [ UITransitionView alloc ]  
initWithFrame:  
CGRectMake(viewRect.origin.x, viewRect.origin.y + 48.0,  
viewRect.size.width, viewRect.size.height - 48.0  
)];
```

После того как переход создан, он добавляется в вид таким же способом, что и другие объекты вида, в качестве подуровня:

```
[ self addSubview: transitionView ];
```

Вызов перехода

Чтобы выполнить переход, вызовите метод `transition` класса. Вы передаете в него стиль перехода и указатели на два вида, между которыми осуществляется переход:

```
[ transitionView transition: 0  
fromView: myOldView  
toView: myNewView  
];
```

Иначе переход может быть вызван путем передачи только нового вида, без передачи старого. Но имейте в виду, что это работает только с некоторыми

переходами. Те виды, которые не поддерживают такой способ, не будут использовать никакого перехода:

```
[ transitionView transition: 0 toView: myNewView ];
```

UIKit на сегодняшний день поддерживает 10 различных переходов. Единичный переходный вид может вызываться неоднократно с различными стилями, перечисленными в табл. 3.3. Это позволяет разработчикам выбирать наиболее подходящий переход в зависимости от конкретного действия, не беспокоясь о создании нового объекта для каждого возможного перехода, который они захотят выполнить.

Таблица 3.3

Стиль	Описание
0	Нет перехода/быстрый переход
1	Страницы прокручиваются влево
2	Страницы прокручиваются вправо
3	Страницы прокручиваются вверх
4	Старая страница отгибается вверх, новая страница отгибается вниз
5	Старая страница отгибается вниз, новая страница отгибается вверх
6	Новая страница постепенно появляется поверх старой
7	Страницы прокручиваются вниз
8	Новая страница отгибается поверх старой
9	Новая страница отгибается вниз поверх старой

Пример: переворачивание страниц

Самый лучший пример использования перехода страниц — это иллюстрация чтения книги. В этом примере 10 страниц текста создаются с помощью объекта UITextView, описанного ранее в этой главе. С помощью панели навигации пользователю предоставляются две кнопки для перемещения между предыдущей и следующей страницами. В зависимости от выбранного пользователем направления используются различные переходы. Когда пользователь достигает конца книги, соответствующая кнопка перемещения становится недоступной.

Листинги 3.9 и 3.10 могут быть скомпилированы с использованием пакета инструментов с помощью следующей командной строки:

```
$ arm-apple-darwin9-gcc -o MyExample MyExample.m -lobjc \
  -framework CoreFoundation -framework Foundation -framework UIKit
```

Листинг 3.9. Пример переворачивания страниц (MyExample.h)

```
#import <CoreFoundation/CoreFoundation.h>
#import <UIKit/UIKit.h>
#import <UIKit/UINavigationController.h>
#import <UIKit/UINavigationControllerItem.h>
#import <UIKit/UITransitionView.h>
#import <UIKit/UITextView.h>

#define MAX_PAGES 10

@interface MainView : UIView
{
    UINavigationController *navBar;      /* Наша панель навигации */
    UINavigationControllerItem *navItem; /* Заголовок панели навигации */
    UITransitionView *transView;      /* Наш переход */
    int pageNum;                      /* Номер текущей страницы */

    /* Несколько страниц для прокручивания */
    UITextView *textPage[MAX_PAGES];
}

- (id)initWithFrame:(CGRect)frame;
- (void)dealloc;
- (UINavigationController *)createNavBar:(CGRect)rect;
- (void)setNavBar;
- (void)navigationBar:(UINavigationController *)navbar
  buttonClicked:(int)button;
- (void)flipTo:(int)page;

@end
```

```
@interface MyApp : UIApplication
{
    UIWindow *window;
    MainView *mainView;
}

- (void)applicationDidFinishLaunching:(NSNotification *)aNotification;
@end
```

Листинг 3.10. Пример переворачивания страниц (MyExample.m)

```
#import "MyExample.h"

int main(int argc, char **argv)
{
    NSAutoreleasePool *autoreleasePool = [
        [ NSAutoreleasePool alloc ] init
    ];
    int returnCode = UIApplicationMain(argc, argv, @"MyApp", @"MyApp");
    [ autoreleasePool release ];
    return returnCode;
}

@implementation MyApp

- (void)applicationDidFinishLaunching:(NSNotification *)aNotification {
    window = [ [ UIWindow alloc ] initWithContentRect:
        [ UIHardware fullScreenApplicationContentRect ]
    ];

    CGRect rect = [ UIHardware fullScreenApplicationContentRect ];
    rect.origin.x = rect.origin.y = 0.0f;

    mainView = [ [ MainView alloc ] initWithFrame: rect ];
}
```

```
[ window setContentView: mainView ];
[ window orderFront: self ];
[ window makeKey: self ];
[ window _setHidden: NO ];
}
@end

@implementation MainView
- (id)initWithFrame:(CGRect)rect {
    self = [ super initWithFrame: rect ];
    if (nil != self) {

        CGRect viewRect;
        int i;

        /* Создаем новый порт вида под панелью навигации */
        viewRect = CGRectMake(rect.origin.x, rect.origin.y + 48.0,
                               rect.size.width, rect.size.height - 48.0);

        /* Устанавливаем нашу начальную страницу */
        pageNum = MAX_PAGES / 2;

        /* Создаем десять объектов UITextView в качестве страниц
         в нашей книге */
        for(i=0;i<MAX_PAGES;i++) {
            textPage[i] = [ [ UITextView alloc ]
                               initWithFrame: rect ];
            [ textPage[i] setText: [ [ NSString alloc ]
                                       initWithFormat: @"Some text for page %d", i+1 ] ];
        }

        /* Создаем панель навигации с кнопками Prev и Next */
        navBar = [ self createNavBar: rect ];
        [ self setNavBar ];
        [ self addSubview: navBar ];
    }
}
```

```
/* Создаем наш переходный вид */
transView = [ [ UITransitionView alloc ]
               initWithFrame: viewRect ];
[ self addSubview: transView ];

/* Переход к первой странице */
[ self flipTo: pageNum ];
}
return self;
}

- (void)dealloc
{
    [ navBar release ];
    [ navItem release ];
    [ self dealloc ];
    [ super dealloc ];
}

- (UINavigationController *)createNavBar:(CGRect)rect {
    UINavigationController *newNav = [ [UINavigationController alloc]
                                       initWithFrame:
                                           CGRectMake(rect.origin.x, rect.origin.y, rect.size.width, 48.0)
                                       ];

    [ newNav setDelegate: self ];
    [ newNav enableAnimation ];

    /* Добавляем наш заголовок */
    navItem = [ [UINavigationControllerItem alloc]
                 initWithTitle:@"My Example" ];
    [ newNav pushViewController: navItem ];

    [ newNav showLeftButton:@"Prev" withStyle: 0
      rightButton:@"Next" withStyle: 0 ];
}
```

```
return newNav;
}

- (void)setNavBar
{

    /* Делаем доступными или недоступными наши кнопки страниц */
    if (pageNum == 1)
        [ navBar setButton: 1 enabled: NO ];
    else
        [ navBar setButton: 1 enabled: YES ];

    if (pageNum == MAX_PAGES)
        [ navBar setButton: 0 enabled: NO ];
    else
        [ navBar setButton: 0 enabled: YES ];
}

- (void)navigationBar:(UINavigationController *)navbar buttonClicked:(int)button
{
    /* Следующая страница */
    if (button == 0)
    {
        [ self flipTo: pageNum+1 ];
    }

    /* Предыдущая страница */
    else {
        [ self flipTo: pageNum-1 ];
    }
}

- (void)flipTo:(int)page {
    int transitionNum; /* Какой номер перехода должен быть использован? */
```

```
if (page < pageNum)
    transitionNum = 2;
else if (page > pageNum)
    transitionNum = 1;
else
    transitionNum = 0;

[ transView transition: transitionNum
  fromView: textPage[pageNum-1] toView: textPage[page-1] ];

pageNum = page;

[ self setNavBar ];
}
@end
```

Как это работает

Вот как работает пример с переворачиванием страниц:

1. При порождении приложения создается объект `MainView` и вызывается его метод `initWithFrame`. Тем самым создаются десять объектов `UITextView`, которые станут примерами страниц книги, а затем он создает панель навигации и один объект перехода. Наконец, вызывается образец метода `flipTo`, который отвечает за переворачивание страниц до указанного номера страницы.
2. Метод `flipTo` выбирает стиль перехода, основываясь на том, является ли страница, к которой надо перейти, предыдущей или последующей.
3. После определения того, какой переход использовать, метод `flipTo` вызывает переходный вид, чтобы проделать всю работу по перемещению к требуемой странице. Затем он устанавливает действующий номер страницы, являющийся переменной в классе.
4. Когда пользователь нажимает кнопку перехода `Prev` или кнопку перехода `Next`, то вызывается метод `buttonClicked` с указателем на панель навигации и номер нажатой кнопки. Здесь снова вызывается метод `flipTo` для перехода к новой странице и для того, чтобы сделать нажатую кнопку перехода недоступной, в случае, если достигнут конец книги.

Для дальнейшего изучения

Вот несколько весьма полезных моментов, которые можно реализовать с помощью примера с переворачиванием страниц.

- ❑ Поэкспериментируйте с различными доступными стилями переходов и придумайте действия, при которых вы хотели бы использовать каждый из переходов.
- ❑ Воспользуйтесь примером с переворачиванием страниц и заполните каждое поле с рамкой произвольным набором символов ASCII. Теперь напишите быстрый переход через все 10 страниц, чтобы создать анимацию ASCII, которая подражает старинному кинопроектору.
- ❑ Проверьте наличие прототипов `UITransitionView.h` в каталоге вашего пакета инструментов `include`. Вы найдете их в папке `/toolchain/sys/usr/include/UIKit/`.

Листы действий

iPhone — относительно небольшое устройство с достаточно ограниченным пространством на экране, и без пишущего пера. В связи с этим пользователю весьма непросто нащупывать кнопки пальцами, и он может случайно нажать не те кнопки. Если такое происходит, то продуманно написанное приложение запрашивает у пользователя подтверждение, прежде чем просто удалить важные данные. Для привлечения к себе внимания приложения на настольном компьютере отображают всплывающие окна. На iPhone же снизу всплывают листы действий (action sheets), затеняя при этом оставшуюся часть экрана до тех пор, пока пользователь не выберет нужный вариант действий. Термин *лист* (sheet) продолжает метафору страниц, используемую Apple для iPhone.

Создание листа действий

Лист действий (action sheet) — это объект, который может быть порожден поверх любого вида, а единичный вид может хранить целый ряд различных листов действий. Основной лист действий состоит из заголовка, текста тела и тех вариантов действий, среди которых должен осуществлять свой выбор пользователь. Он порождается точно так же, как и другие объекты `UIView` с помощью метода `initWithFrame`. Поскольку листы предупреждений появля-

ются внизу, то координаты окна можно отсчитывать вниз от половины экрана ($Y = 240$):

```
UIActionSheet *actionSheet = [ [ UIActionSheet alloc ]
    initWithFrame: CGRectMake(0, 240, 320, 240) ];
[ actionSheet setTitle:@"Computer doesn't like people" ];
[ actionSheet setBodyText: [ NSString stringWithFormat:
    @"I did not complete your request because I don't like humans." ]
];
[ actionSheet setDelegate: self ];
```

Листы действий могут растянуть рамку, чтобы вместить все элементы, которые необходимо отобразить, поэтому статического прямоугольника 320×240 должно быть достаточно для стандартных листов действий.

Как и панели навигации, листы действий могут поддерживать один из трех стилей (табл. 3.4).

Таблица 3.4

Стиль	Описание
0	Стиль по умолчанию, серый цвет фона, белый цвет текста
1	Сплошной черный цвет, белый цвет текста
2	Прозрачный черный цвет фона, белый цвет текста

Стиль может быть задан с помощью вызова `setActionSheetStyle:`

```
[ actionSheet setActionSheetStyle: 0 ];
```

Кнопки листа действий

В редких случаях имеет смысл отображать лист действий с кнопками, например, для отображения индикатора выполнения (описанного в главе 7). Однако в большинстве случаев листы предупреждения выводятся только с целью напоминания пользователю о чем-либо. Листы предупреждения могут вмещать в себя столько кнопок, сколько их может поместиться на экране. Чтобы добавить кнопку, вызовите метод `addButtonWithTitle` листа действий:

```
[ actionSheet addButtonWithTitle:@"OK" ];
```

Кнопки уничтожения

Кнопки, подтверждающие полное удаление или какое-либо другое действие, которое может привести к уничтожению данных, должны использовать то, что в API называется *кнопкой уничтожения* (destructive button). Кнопки уничтожения отображаются ярко-красным цветом, чтобы предупредить пользователя о том, что он собирается выполнить какое-либо важное действие, которое невозможно будет отменить.

Метод `setDestructiveButtonIndex` используется для того, чтобы пометить кнопку как кнопку уничтожения. Он принимает на вход порядковый номер кнопки:

```
[ actionSheet addButtonWithTitle:@"Confirm Delete" ];
[ actionSheet setDestructiveButtonIndex: 0 ];
```

Отображение листа действий

После того как вы определитесь с текстом и кнопками, которые нужно отобразить, вы уже можете вывести лист пользователю. В зависимости от желаемого эффекта можно вызвать пять различных методов. Каждый из этих методов принимает класс, производный от `UIView`, который должен быть указателем на класс того вида, который задействован листом действий:

```
[ actionSheet presentSheetFromAboveView: myView ];
[ actionSheet presentSheetFromBehindView: myView ];
[ actionSheet presentSheetFromBarButton: buttonBar ];
[ actionSheet presentSheetInView: myView ];
[ actionSheet presentSheetToAboveView: myView ];
```

Наиболее распространенный способ вызова листа действий — это с помощью `presentSheetInView`, который может быть вызван из вида, используя в качестве аргумента `self`.

Перехват нажатий кнопок

После того как лист действий отображен, элемент управления возвращается в программу. Как было показано ранее в этой главе, многие объекты используют метод обратного вызова под названием `buttonClicked` всякий раз, когда пользователь касается какой-либо кнопки. Это позволяет приложению про-

должать свое выполнение в фоне и прерываться только тогда, когда что-то действительно происходит.

Далее представлен прототип метода обратного вызова для листа действий. Если делегат листа действий настроен на вызываемый вид, то ожидается, что этот вызываемый вид будет отвечать на нажатие кнопки:

```
- (void)actionSheet:(UIActionSheet *)sheet buttonClicked:(int)button;
```

Метод `buttonClicked` вызывается с указателем на выбранный лист и порядковый номер кнопки. Кнопки пронумерованы сверху вниз, начиная с 1 (они нумеруются не с 0, как другие значения):

```
- (void)actionSheet:(UIActionSheet *)sheet buttonClicked:(int)button {
    if (sheet == actionSheet) {
        switch(button) {
            case 1:
                /* Нажата самая верхняя кнопка */
                break;
            case 2:
                /* Нажата вторая сверху кнопка */
                break;
        }
    }
}
```

Отмена листа действий

Наконец, после обработки нажатия кнопки лист действий должен исчезнуть с экрана, если только, конечно же, приложению не требуется, чтобы пользователь нажал еще и другие кнопки. Чтобы убрать лист с экрана, воспользуйтесь методом `dismiss`:

```
[ sheet dismiss ];
```

Пример: кнопка "End-of-the-World"

Правительство решило, что президенту было бы гораздо удобнее носить с собой iPhone вместо чемоданчика с большой красной кнопкой. Один из их главных программистов без труда написал приложение `End-of-the-World.app`,

которое президент может запустить в любой момент, чтобы пустить в ход ядерное оружие, которое приведет к концу света (или, по крайней мере, к началу передела мира). Однако существует проблема, связанная с возможными множественными случайными нажатиями этой кнопки. С вами заключили контракт на добавление к этому приложению листа подтверждения на случай, если президент на самом деле не хотел, чтобы наступил конец света.

Наше приложение-пример отображает панель навигации с кнопкой **End World**. При нажатии она сначала запросит подтверждение, прежде чем наступит конец света. Полиморфизм будет проиллюстрирован присутствием двух методов `buttonClicked`. При нажатии кнопки панели навигации автоматически будет вызываться корректный метод `buttonClicked`.

Листинги 3.11 и 3.12 могут быть построены с использованием пакета инструментов с помощью следующей командной строки:

```
$ arm-apple-darwin9-gcc -o MyExample MyExample.m -lobjc \
  -framework CoreFoundation -framework Foundation -framework UIKit
```

Листинг 3.11. Пример листа действий (MyExample.h)

```
#import <CoreFoundation/CoreFoundation.h>
#import <UIKit/UIKit.h>
#import <UIKit/UIActionSheet.h>
#import <UIKit/UINavigationController.h>

@interface MainView : UIView
{
    UIActionSheet *endWorldSheet;
    UIActionSheet *deniedSheet;
    UINavigationController *navBar;
}

- (id)initWithFrame:(CGRect) frame;
- (void)dealloc;
@end

@interface MyApp : UIApplication
{
```

```
UIWindow *window;  
MainView *mainView;  
}  
  
- (void)applicationDidFinishLaunching:(NSNotification *)aNotification;  
@end
```

Листинг 3.12. Пример листа действий (MyExample.m)

```
#import "MyExample.h"  
  
int main(int argc, char **argv)  
{  
    NSAutoreleasePool *autoreleasePool = [  
        [ NSAutoreleasePool alloc ] init  
    ];  
    int returnCode = UIApplicationMain(argc, argv, @"MyApp", @"MyApp");  
    [ autoreleasePool release ];  
    return returnCode;  
}  
  
@implementation MyApp  
  
- (void)applicationDidFinishLaunching:(NSNotification *)aNotification {  
    window = [ [ UIWindow alloc ] initWithContentRect:  
        [ UIScreen fullScreenApplicationContentRect ]  
    ];  
  
    CGRect rect = [ UIScreen fullScreenApplicationContentRect ];  
    rect.origin.x = rect.origin.y = 0.0f;  
  
    mainView = [ [ MainView alloc ] initWithFrame: rect ];  
  
    [ window setContentView: mainView ];  
    [ window orderFront: self ];  
}
```

```
[ window makeKey: self ];
[ window _setHidden: NO ];
}
@end

@implementation MainView
- (id)initWithFrame:(CGRect)rect {
    self = [ super initWithFrame: rect ];
    if (nil != self) {

        /* Создаем кнопку конца света */
        navBar = [ [UINavigationController alloc] initWithFrame:
            CGRectMake(rect.origin.x, rect.origin.y, rect.size.width, 48.0f)
        ];
        [ navBar setDelegate: self ];
        [ navBar enableAnimation ];
        [ navBar showLeftButton:nil withStyle: 0
            rightButton:@"End World" withStyle: 3 ];
        [ self addSubview: navBar ];
    }
    return self;
}

- (void)actionSheet:(UIActionSheet *)sheet buttonClicked:(int)button
{
    if (sheet == endWorldSheet) {
        if (button == 1) {

            /* Сюда поместите свой код, реализующий конец света */
            /* TODO: Хотите ли вы отменить лист прежде, чем наступит конец
            света? */
        }
    }
    [ sheet dismiss ];
}
```

```
- (void)navigationBar:(UINavigationController *)navbar buttonClicked:(int)button
{
    /* Спросите о конце света */

    endWorldSheet = [ [ UIActionSheet alloc ] initWithFrame:
        CGRectMake(0, 240, 320, 240)
    ];

    [ endWorldSheet setTitle: @"Please Confirm" ];
    [ endWorldSheet setBodyText:@"I noticed you are trying to end the
world. Are you sure you want to do this?" ];
    [ endWorldSheet addButtonWithTitle:@"End World" ];
    [ endWorldSheet setDestructiveButtonIndex: 0];
    [ endWorldSheet addButtonWithTitle:@"Cancel" ];
    [ endWorldSheet setDelegate: self ];
    [ endWorldSheet presentSheetInView: self ];
}

- (void)dealloc
{
    [ self dealloc ];
    [ super dealloc ];
}

@end
```

Как это работает

Эта программа работает следующим образом:

1. При порождении приложения создается объект `MainView` и вызывается его метод `initWithFrame`. Тем самым создаются и отображаются пользователю вид и панель навигации.
2. Когда пользователь нажимает кнопку **End World**, объект уведомляет метод своего делегата `buttonClicked`. Поскольку ожидается, что объект панели навигации передаст указатель на себя, то вызывается именно метод `buttonClicked`, принимающий параметр `UINavigationController *` (исходя из правил полиморфизма).

3. Метод `buttonClicked` создает лист действий, вызываемый `endWorldSheet`, и представляет его пользователю. Он незамедлительно возвращает управление, но делает содержимое вида затененным и недоступным для пользователя.
4. Когда пользователь нажимает кнопку на листе действий, объект листа предупреждения уведомляет метод своего делегата `buttonClicked`. Поскольку класс листа предупреждения передает параметр `UIActionSheet *`, то вызывается соответствующая версия `buttonClicked`.
5. Метод сравнивает передаваемый в него указатель с указателем на `endWorldSheet`. Если это один и тот же указатель, то метод узнает о том, что в листе нажата кнопка. Затем он смотрит на порядковый номер кнопки и предпринимает соответствующие действия.
6. Если пользователь подтверждает свое решение нажатием кнопки **End World**, то сам метод `buttonClicked` создает и отображает новый лист действий, информирующий пользователя о том, что произошла ошибка.
7. Когда пользователь нажимает кнопку **OK**, то снова вызывается тот же самый метод `buttonClicked`, и только в этот раз передаваемый в него указатель листа действий будет указателем `deniedSheet`. Метод без промедления игнорирует его и просто отменяет лист действий, не предпринимая при этом каких-либо дальнейших действий.

Для дальнейшего изучения

Протестируйте листы действий, чтобы лучше понять, как они работают.

- Поэкспериментируйте с добавлением различных кнопок в лист действий. Сколько кнопок поместиться на экране? Какой длины текст они смогут уместить?
- Создайте лист действий без кнопок, который бы информировал пользователя о том, что загружается какой-либо файл. Используйте `NSTimer`, чтобы подождать 10 секунд, а затем отмените лист, вообще не применяя метод `buttonClicked`.
- Проверьте наличие прототипов `UIActionSheet.h` в каталоге вашего пакета инструментов `include`. Вы найдете их в папке `/toolchain/sys/usr/include/UIKit/`. Поэкспериментируйте с некоторыми менее документированными методами.

Таблицы

Таблицы являются базой для большинства списков выбора в iPhone. Голосовая почта, последние звонки и даже электронная почта — все они используют богатые функциональные возможности класса `UITableView` для отображения своих списков элементов. Помимо того что класс `UITableView` является простым инструментом выбора из списка, он еще и содержит встроенную функциональность добавления раскрытий, проведения для удаления, анимации, меток и даже изображений.

Создание таблиц

Таблица (`table`) имеет три основных составляющих: сама таблица, столбцы таблицы и ячейки таблицы. Данные таблицы помещаются в очередь из *привязки данных* (`data binding`) таблицы. Привязка данных — это интерфейс, используемый таблицей для помещения в очередь информации о том, какие данные выводить на экран, например, имена файлов, сообщения электронной почты и т. д. *Источник данных* (`data source`) — это объект, отвечающий за эту очередь. После того как таблица будет создана, к ней должен быть прикреплен источник данных, чтобы таблица могла что-либо отображать. Он вызывается всякий раз, когда таблица перезагружается, или же путем прокручивания в область просмотра попадают новые ячейки, и сообщает таблице, какие столбцы и строки с данными в них необходимо отобразить.

Наследование `UITableView`

Таблица как свой собственный источник данных вполне подходит для большинства специфических применений. Это позволяет классу таблицы и данным таблицы быть аккуратно обернутыми в один класс. Чтобы это сделать, унаследуйте объект `UITableView` для создания нового класса для ваших данных. В приведенном далее примере создается подкласс под названием `MyTable`. Для обеспечения функциональности части класса, касающейся таблицы, подменяются методы базового класса, используемые для инициализации и разрушения объекта:

```
@interface MyTable : UITableView
{
}
```

```
-(id)initWithFrame:(struct CGRect)rect;
-(void) dealloc;
```

Чтобы добавить часть класса, касающуюся источника данных, используются два метода, чтобы заставить привязку данных таблицы загрузить данные: `numberOfRowsInTable` и `cellForRow`. Поскольку таблица выступает в роли своего собственного источника данных, то вы должны написать эти методы в вашем подклассе, где методы будут отвечать за возврат данных столбцов и строк для таблицы.

```
-(int)numberOfRowsInTable:(UITableView *)_table;
-(UITableViewCell *)table:(UITableView *)table
cellForRow:(int)row
column:(UITableViewColumn *)col;
```

Мы рассмотрим эти методы в разд. "Привязка данных" далее в этой главе.

Подмена методов `UITableView`

После создания подкласса `UITableView` должны быть подменены методы инициализации и уничтожения. Это позволит подклассу при создании определить собственные столбцы и стиль и корректно освободить все ресурсы, созданные им для себя.

Методом инициализации для объекта `UITableView` является `initWithFrame`:

```
-(id)initWithFrame:(struct CGRect)rect {
    self = [super initWithFrame: rect];
    if (nil != self) {
        /* Добавьте сюда дополнительный код инициализации */
    }
}
```

Поскольку таблица выступает в роли собственного источника данных, то можно использовать метод `initWithFrame` для определения столбцов таблицы. Столбец создается как класс `UITableViewColumn` и имеет внутри таблицы свой заголовок, идентификатор и ширину. Объект `UITableViewColumn` также используется многими производными классами, например, сортировщиками, обсуждаемыми в главе 7:

```
UITableViewColumn *myColumn = [ [ UITableViewColumn alloc ]
initWithTitle: @"Column 1"
identifier:@"column1"
```

```

width: rect.size.width
];
[ self addTableColumn: myColumn ];

```

Чтобы создать замкнутый класс таблицы, объект `UITable` может быть назначен в качестве своего собственного источника данных. Чтобы привязать `self` как источник данных, выполните `setDataSource:`:

```
[ self setDataSource: self ];
```

Подмените метод `dealloc` так, чтобы вы могли добавить код для освобождения всех ресурсов, которые должны освободиться при уничтожении объекта. Когда объект `MyTable` уничтожен, метод `dealloc` должен освободить его столбцы и любые другие ресурсы, которые он занимал. Кроме того, потребуется вызвать метод `dealloc` его суперкласса, чтобы освободить все ресурсы, созданные внутренне классом `UITable`:

```

- (void)dealloc {
    [ myColumn release ];
    [ super dealloc ];
}

```

Привязка данных

Как было упомянуто в разд. "Наследование `UITable`" ранее в этой главе, привязка данных для `UITable` состоит из двух методов, предоставляющих данные строк для таблицы. Метод `numberOfRowsInTable` просто возвращает целое число, отражающее количество строк данных в таблице. Значение используется объектом таблицы для задания количества строк:

```

- (int)numberOfRowsInTable:(UITable *)_table {
    return 3;
}

```

Второй метод, `cellForRow`, возвращает отдельные строки из таблицы. Он вызывается для каждой строки всякий раз, когда строка попадает в поле зрения. Отдельные строки порождаются от класса `UITableCell`. Приведенный далее пример определяет метод `cellForRow`, создающий ячейку из класса `UITableCell`, определяет ее заголовок, исходя из номера строки, и возвращает ячейку:

```

- (UITableCell *)table:(UITable *)table
cellForRow:(int)row

```

```
column:(UITableColumn *)col
{
    UITableViewCell *cell = [ [ UITableViewCell alloc ] init ];
    NSString *title;

    title = [ [ NSString alloc ] initWithFormat: @"Row %d", row ];
    [ cell setTitle: title ];
    return [ cell autorelease ];
}
```

Простейшая ячейка — это чисто текстовая ячейка, содержащая заголовок, разделитель и необязательное раскрытие (кратко уже описанное). Поскольку эти свойства задаются для каждой ячейки строки, то любая ячейка в таблице может быть отформатирована таким образом, чтобы удовлетворить содержащимся в ней данным.

Метки

Метки (labels) являются миниатюрными классами видов, которые могут быть добавлены к ячейкам таблицы, чтобы добавить ячейке таблицы еще больше декорирующего текста. Для различных целей существуют различные классы меток. Например, метка Web-вида имеет вид серого прозрачного овала, содержащего текст. В приведенном далее примере создается класс `UIWebViewLabel` с такими отступами, чтобы он отображался в левом верхнем углу ячейки:

```
UIWebViewLabel *label = [ [ UIWebViewLabel alloc ] initWithFrame:
    CGRectMake(0.0, 3.0f, 320.0, 20.0) ];
[ label setText: @"My UIWebView" ];
[ cell addSubview: label ];
```

Доступны следующие классы меток (табл. 3.5).

Таблица 3.5

Класс	Описание
<code>UITextLabel</code>	Отображает простой текст в области просмотра
<code>UITextLabelField</code>	Предоставляет окно ввода текста

Таблица 3.5 (окончание)

Класс	Описание
UIDateLabel	Отображает дату в ее области просмотра
UIWebViewLabel	Отображает текст на серой прозрачной поверхности

Раскрытия

Раскрытия (disclosures) — это значки, отображаемые в правой части ячейки таблицы, чтобы указать на то, что там существует еще один уровень информации, который отображается при выделении ячейки. Это весьма распространено на настольных компьютерах в интерфейсах, подобных iTunes, где пользователь сначала выбирает жанр, затем исполнителя и в конце — песню.

Чтобы разрешить применение раскрытия для конкретной ячейки таблицы, воспользуйтесь методом `setShowDisclosure:`:

```
[ cell setShowDisclosure: YES ];
```

Стиль раскрытия также может быть изменен:

```
[ cell setDisclosureStyle: 0 ];
```

Доступны следующие стили раскрытия (табл. 3.6).

Таблица 3.6

Стиль	Описание
0	Черная стрелка
1	Синий кружок с белой стрелкой

Ячейки с изображениями и текстовые ячейки

Рядом с текстом строк таблицы могут отображать изображения. Это требует использования различных типов ячеек таблицы под названием `UIImageAndTextTableCell`. В примере класса `MyTable` в методе `cellForRow` должен возвращаться этот тип объекта вместо `UISimpleTableCell`:

```
UIImageAndTextTableCell *cell =
    [ [ UIImageAndTextTableCell alloc ] init ];
```

```

UIImageView *image = [ [ UIImage alloc ]
    initWithContentsOfFile: @"/path/to/file.png" ];
[ cell setTitle: @"My row, now with image!" ];
[ cell setImage: image ];
return [ cell autorelease ];

```

Для загрузки файла изображения используется класс `UIImage`. Этот класс подробно рассмотрен в *главе 7*.

Этот тип ячейки данных позволяет также с помощью метода `setAlignment` менять выравнивание изображения и текста:

```
[ cell setAlignment: 2 ];
```

Поддерживаются следующие стили выравнивания (табл. 3.7).

Таблица 3.7

Стиль	Описание
0	Изображение и текст выравниваются по левому краю с разделителем поля
1	Изображение и текст выравниваются по левому краю без разделителя поля
2	Изображение выравнивается по левому краю, текст — по центру
3	Изображение выравнивается по левому краю, текст — по правому

В зависимости от размера загружаемого изображения может потребоваться изменение высоты строки, чтобы вместить все изображение. Метод `setRowHeight` является частью базового класса `UITableView` и может быть задан при инициализации таблицы:

```
[ self setRowHeight: 64 ];
```

Поскольку эти настройки — настройки уровня таблицы, то заданная высота строки повлияет на все ячейки одинаково.

Провести, чтобы удалить

Объект `UITableView` обладает встроенной логикой перехватывания проводимых касаний и отображения подтверждений на удаление. Это позволяет пользователю проводить пальцами по строке, которую он хочет удалить.

Чтобы перехватить удаляющие касания, подмените метод `swipe` в подклассе, порожденном от `UITableView`:

```
- (int)swipe:(int)type withEvent:(struct _GSEvent *)event;
{
    CGPoint point = GSEventGetLocationInWindow(event);
    CGPoint offset = _startOffset;

    point.x += offset.x;
    point.y += offset.y;
    int row = [ self rowAtPoint:point ];

    [ [ self visibleCellForRow: row column:0 ]
      _showDeleteOrInsertion: YES
      withDisclosure: NO
      animated: YES
      isDelete: YES
      andRemoveConfirmation: YES
    ];

    return [ super swipe:type withEvent:event ];
}
```

В метод `swipe` передается событие касания, которое обрабатывается платформой `Graphics Services` (см. главу 4). Вызов метода `GSEventGetLocationInWindow` этой платформы определяет точку на экране, в которой произошло касание. Поскольку возвращается точка на экране (а не в рамках окна), то во внимание необходимо принимать смещение положения окна на экране. Например, если окно отображается ниже 48-точечной панели навигации, то необходимо учитывать это при вертикальном смещении. В данном примере для пересчета координат экрана в координаты окна используется переменная `_startOffset`.

После того как координаты области касания будут определены, вычисляется количество строк путем передачи этих координат методу `rowAtPoint`, принадлежащему классу `UITableView`. Он возвращает количество строк в виде целого числа.

Далее, с помощью метода `visibleCellForRow` класса `UITableView` определяется сама ячейка таблицы. Он возвращает указатель на выбранную ячейку табли-

цы, чей метод `_showDeleteOrInsertion` может быть вызван для отображения подтверждения на удаление. Оно имеет вид кнопки удаления красного цвета.

Чтобы запретить пользователю удалять строки из таблицы, этот метод должен просто возвращать, не предпринимая никаких действий.

Когда пользователь подтвердит, что хочет удалить строку (путем нажатия кнопки удаления), вызывается внутренний метод `_willDeleteRow`. Если какая-либо строка удалена, то эта строка должна быть удалена еще и из источника данных, иначе она будет отображена снова после того, как пользователь прокрутит ее на экране. Кроме того, могут быть еще и дополнительные операции, которые должно выполнить приложение, например, удаление файла или сообщения, связанного с ячейкой:

```
- (void)_willDeleteRow:(int)row
forTableCell:(id)cell
viaEdge:(int)edge
animateOthers:(BOOL)animate
{
    /* Здесь выполните любые операции удаления */

    [fileList removeObjectAtIndex:row];
    [super _willDeleteRow:row
     forTableCell:cell
     viaEdge:edge
     animateOthers:animate];
};
```

Выбор элемента

Когда пользователь выделяет какую-либо ячейку таблицы, вызывается метод таблицы `tableView:tableViewSelected`. Данный метод должен быть подменен, чтобы обрабатывать выделение, например, при открытии файла или запуске приложения. Количество строк можно получить с помощью метода `tableView:selectedRow` базового класса:

```
- (void)tableViewSelected:(NSNotification *)notification
{
    int index = [self tableView:selectedRow];
};
```

```

/* Был выбран файл. Здесь сделайте с этим что-нибудь. */
}

```

После того как найден порядковый номер строки, как показано выше, вы можете использовать его для ссылки на действительное значение строки из ваших собственных данных и произвести над ним соответствующие действия. Например, если ваша таблица является массивом файлов, то вы будете использовать порядковый номер для ссылки на корректное имя файла в вашем массиве. И тогда вы сможете загрузить файл или выполнить какие-либо другие операции, ради которых ваше приложение и задумывалось.

Пример: проводник файлов

В этом примере мы создадим собственный класс `UITable` для общего использования в качестве проводника файлов. Данный проводник будет читать каталог, передаваемый с помощью метода `setPath`, и отображать все имеющиеся в нем файлы с расширением, также передаваемым с помощью метода `setExtension`. Для большей наглядности примера была добавлена функциональность удаления, но чтобы предотвратить ошибочное удаление файлов в процессе экспериментирования с данным примером, она удаляет файлы только из списка, а не из файловой системы iPhone.

Данный пример может быть легко добавлен к любому приложению iPhone для отображения списка файлов путем включения следующего кода в класс вида приложения:

```

#import "FileTable.h"

FileTable *fileTable = [ [ FileTable alloc ] initWithFrame: rect ];
[ fileTable setPath: @"/Applications" ];
[ fileTable setExtension: @"app" ];
[ fileTable reloadData ];
[ self addSubview: fileTable ];

```

Полное приложение, показанное в листингах 3.13—3.18, может быть скомпилировано из пакета инструментов с помощью следующей командной строки:

```

$ arm-apple-darwin9-gcc -o MyExample MyExample.m FileTable.m \
DeletableCell.m \
-lobjc -framework CoreFoundation -framework Foundation -framework UIKit

```

Листинг 3.13. Заголовок класса приложения (MyExample.h)

```
#import <CoreFoundation/CoreFoundation.h>
#import <UIKit/UIKit.h>
#import "FileTable.h"

@interface MainView : UIView
{
    FileTable *fileTable;
}
- (id)initWithFrame:(CGRect)frame;
- (void)dealloc;
@end

@interface MyApp : UIApplication
{
    UIWindow *window;
    MainView *mainView;
}
- (void)applicationDidFinishLaunching:(NSNotification *)aNotification;
@end
```

Листинг 3.14. Реализация класса приложения (MyExample.m)

```
#import "MyExample.h"

int main(int argc, char **argv)
{
    NSAutoreleasePool *autoreleasePool =
        [ [ NSAutoreleasePool alloc ] init ];
    int returnCode = UIApplicationMain(argc, argv, @"MyApp", @"MyApp");
    [ autoreleasePool release ];
    return returnCode;
}

@implementation MyApp
```

```
- (void)applicationDidFinishLaunching:(NSNotification *)aNotification {
    window = [ [ UIWindow alloc ] initWithContentRect:
        [ UIHardware fullScreenApplicationContentRect ]
    ];

    CGRect rect = [ UIHardware fullScreenApplicationContentRect ];
    rect.origin.x = rect.origin.y = 0.0f;

    mainView = [ [ MainView alloc ] initWithFrame: rect ];

    [ window setContentView: mainView ];
    [ window orderFront: self ];
    [ window makeKey: self ];
    [ window _setHidden: NO ];
}
@end

@implementation MainView
- (id)initWithFrame:(CGRect)rect {

    self = [ super initWithFrame: rect ];
    if (nil != self) {
        fileTable = [ [ FileTable alloc ] initWithFrame: rect ];
        [ fileTable setPath: @"/Applications" ];
        [ fileTable setExtension: @"app" ];
        [ fileTable reloadData ];
        [ self addSubview: fileTable ];
    }
    return self;
}

- (void)dealloc
{
    [ self dealloc ];
    [ super dealloc ];
}
@end
```

Листинг 3.15. Заголовки класса выбора файлов (FileTable.h)

```
#import <UIKit/UIKit.h>
#import <UIKit/UISimpleTableCell.h>
#import <UIKit/UIImageAndTextTableCell.h>
#import <UIKit/UITableColumn.h>
#import <UIKit/UIImage.h>
#import <GraphicsServices/GraphicsServices.h>

@interface FileTable : UITableView
{
    NSString *path;
    NSString *extension;
    NSMutableArray *fileList;
    UITableColumn *colFilename;
    UITableColumn *colType;
}

- (id)initWithFrame:(struct CGRect)rect;
- (void)setPath:(NSString *)_path;
- (void)setExtension:(NSString *)_extension;
- (void)reloadData;
- (int)swipe:(int)type withEvent:(struct _GSEvent *)event;
- (int)numberOfRowsInTable:(UITableView *)_table;
- (UITableViewCell *)table:(UITableView *)table
  cellForRow:(int)row
  column:(UITableColumn *)col;
- (void)_willDeleteRow:(int)row forTableCell:(id)cell viaEdge:(int)edge
  animateOthers:(BOOL)animate;
- (void)dealloc;
@end
```

Листинг 3.16. Реализация класса выбора файла (FileTable.m)

```
#import "FileTable.h"
#import "DeletableCell.h"
```

```
@implementation FileTable

- (id)initWithFrame:(struct CGRect)rect {
    self = [ super initWithFrame: rect ];
    if (nil != self) {

        colFilename = [ [ UITableViewColumn alloc ]
            initWithTitle: @"Filename"
            identifier:@"filename"
            width: rect.size.width - 75
        ];
        [ self addTableColumn: colFilename ];

        colType = [ [ UITableViewColumn alloc ]
            initWithTitle: @"Type"
            identifier:@"type"
            width: 75
        ];
        [ self addTableColumn: colType ];

        [ self setSeparatorStyle: 1 ];
        [ self setDelegate: self ];
        [ self setDataSource: self ];
        [ self setRowHeight: 64 ];

        fileList = [ [ NSMutableArray alloc] init ];
    }

    return self;
}

- (void) setPath:(NSString *)_path {
    path = [ _path copy ];
}
}
```

```
- (void) setExtension:(NSString *)_extension {
    extension = [ _extension copy ];
}

- (void) reloadData {
    NSFileManager *fileManager = [ NSFileManager defaultManager ];
    NSDirectoryEnumerator *dirEnum;
    NSString *file;

    if ([ fileManager fileExistsAtPath: path ] == NO) {
        return;
    }

    [ fileList removeAllObjects ];

    dirEnum = [ [ NSFileManager defaultManager ] enumeratorAtPath: path ];
    while ((file = [ dirEnum nextObject ])) {
        if ([ file hasSuffix: extension ] == YES) {
            [ fileList addObject: file ];
        }
    }

    [ super reloadData ];
}

- (int)numberOfRowsInTable:(UITableView *)_table {
    return [ fileList count ];
}

- (UITableViewCell *)table:(UITableView *)table
cellForRow:(int)row
column:(UITableViewController *)col

    if (col == colFilename) {
        DeletableCell *cell = [ [ DeletableCell alloc ] init ];
    }
}
```

```
[ cell setTable: self ];
UIImageView *image = [ [ UIImageView alloc ]
initWithContentsOfFile:
[ [ NSString alloc ]
initWithFormat: @"/Applications/%@/icon.png",
[ fileList objectAtIndex: row ] ] ];
[ cell setTitle: [ [ fileList objectAtIndex: row ]
stringByDeletingPathExtension ] ];
[ cell setImage: image ];
[ cell setShowDisclosure: YES ];
[ cell setDisclosureStyle: 3 ];
return [ cell autorelease ];
} else if (col == colType) {
DeletableCell *cell = [ [ DeletableCell alloc ] init ];
[ cell setTable: self ];
[ cell setTitle: extension ];
return [ cell autorelease ];
}
}

- (int)swipe:(int)type withEvent:(struct _GSEvent *)event;
{
CGPoint point= GSEventGetLocationInWindow(event);
CGPoint offset = _startOffset;

if (point.x < 100 || point.x > 200) {
point.x += offset.x;
point.y += offset.y;
int row = [ self rowAtPoint:point ];

[ [ self visibleCellForRow: row column: 1 ]
_showDeleteOrInsertion: YES
withDisclosure: NO
animated: YES
isDelete: YES
```

```
        andRemoveConfirmation: YES
    ];

    return [ super swipe:type withEvent:event ];
}

- (void)_willDeleteRow:(int)row
  forTableCell:(id)cell
  viaEdge:(int)edge
  animateOthers:(BOOL)animate
{
    [ fileList removeObjectAtIndex: row ];
    [ super _willDeleteRow: row forTableCell: cell viaEdge: edge
      animateOthers: animate ];
}

- (void)tableViewSelected:(NSNotification *)notification {
    NSString *fileName = [ fileList objectAtIndex: [ self selectedRow ] ];
    /* Был выбран файл. Здесь сделайте с этим что-либо. */
}

- (void)dealloc {
    [ colFilename release ];
    [ colType release ];
    [ fileList release ];
    [ super dealloc ];
}

@end
```

Листинг 3.17. Заголовок класса удаляемой ячейки (DeletableCell.h)

```
#import "FileTable.h"

@interface DeletableCell : UIImageAndTextTableCell
{
```

```
FileTable *table;
}
- (void)setTable:(FileTable *)_table;

@end
```

Листинг 3.18. Реализация класса удаляемой ячейки (DeletableCell.m)

```
#import "DeletableCell.h"

@implementation DeletableCell

- (void)removeControlWillHideRemoveConfirmation:(id)fp8
{
    [ self _showDeleteOrInsertion:NO
      withDisclosure:NO
      animated:YES
      isDelete:YES
      andRemoveConfirmation:YES
    ];
}

- (void)_willBeDeleted
{
    int row = [ table _rowForTableCell: self ];

    /* Сделайте что-нибудь; эта строка удаляется. */
}

- (void)setTable:(FileTable *)_table {
    table = _table;
}

@end
```

Как это работает

Класс `FileTable` работает следующим образом:

1. Когда вызывающий класс размещает новый объект `FileTable`, то он вызывает его метод `initWithFrame`.
2. Это приводит к созданию двух объектов `UITableColumn`, которые становятся столбцами таблицы: `Filename` и `Type`. Этот класс также определяет себя как источник данных и задает некоторые эстетические свойства.
3. Методы таблицы файлов `setPath` и `setExtension` используются для задания нужного пути и расширения отображаемых файлов.
4. Вызывающий класс вызывает метод `reloadData` объекта. Это приводит к тому, что таблица файлов генерирует список файлов с указанным расширением в каталоге по указанному пути. Этот список хранится в массиве `fileList`. Затем вызывается метод `reloadData` суперкласса, который посредством метода `numberOfRowsInTable` ставит в очередь количество строк из источника данных.
5. Как только таблица файлов готова к отображению ячеек, она снова вызывает свою привязку данных, чтобы получить данные для каждой ячейки в виде. Для каждого столбца и каждой строки в виде вызывается метод `cellForRow`, который возвращает соответствующие запросу ячейки строки.
6. Если пользователь проводил пальцем вдоль строки, то вызывается метод `swipe` класса. В результате отображается кнопка подтверждения на удаление путем вызова метода `visibleCellForRow` с соответствующими свойствами.
7. Если пользователь подтверждает удаление, то вызывается метод `_willDeleteRow` ячейки таблицы. В результате строка удаляется из источника данных, поэтому она больше не будет отображаться на экране, даже если ячейку прокрутить на экране.

Для дальнейшего изучения

Чтобы проверить глубину вашего знакомства с таблицами, попробуйте выполнить следующие упражнения.

- Встройте эту таблицу в пример "Hello, World!", приведенный в начале главы. Вместо отображения объекта `UITextView` постройте объект `FileTable` из последнего примера. Убедитесь в том, что при компиляции

вы импортировали (`#import`) заголовок класса и включили его в исходный список.

- Измените данный пример, добавив метод, разрешающий или запрещающий удаления. Это должно изменить поведение метода `swipe` так, что при запрещенном удалении он станет преждевременно возвращать.
- Используйте этот пример для отображения приложений в папке `/Applications`. Измените этот пример, чтобы использовать объект `UIImageAndTextTableCell` для отображения значка приложения рядом с его именем.
- Проверьте наличие прототипов `UITableView.h`, `UITableViewCell.h`, `UIImageAndTextTableCell.h` и `UITableViewColumn.h` в каталоге вашего пакета инструментов. Вы найдете их в папке `/toolchain/sys/usr/include/UIKit/`.
- Проверьте различные типы меток. Прототипы `UITextFieldLabel.h`, `UILabel.h`, `UIDateLabel.h` и `UIWebViewLabel.h` можно найти в каталоге вашего пакета инструментов `/toolchain/sys/usr/include/UIKit/`.

Манипуляции строкой состояния

Внешний вид строки состояния можно настраивать, чтобы она соответствовала духу и назначению вашего приложения, а также строка состояния может отображать уведомления о вашем приложении. Например, когда на iPod фоне воспроизводится музыка, то приложение iPod отображает в строке состояния треугольный значок воспроизведения музыки. Когда включен будильник, приложение будильника отображает в строке состояния небольшие часы. Многие свойства строки состояния можно менять с помощью классов `UIApplication` и `UIHardware`.

Режим строки состояния

Режим строки состояния определяет ее цвет, прозрачность и ориентацию. Кроме того, он может использоваться для анимации строки состояния при переходе между различными режимами. Чтобы задать режим, используйте один из четырех методов `setStatusBarItemMode` в рамках экземпляра вашего приложения, объекта `UIApplication`:

- `(void)setStatusBarMode:(int)mode duration:(float)duration`
- `(void)setStatusBarMode:(int)mode orientation:(int)orientation duration:(float)duration`

```

- (void)setStatusBarMode:(int)mode orientation:(int)orientation
  duration:(float)duration fenceID:(int)fenceID
- (void)setStatusBarMode:(int)mode orientation:(int)orientation
  duration:(float)duration fenceID:(int)fenceID
  animation:(int)animation;

```

Режим

Режим (mode) строки состояния определяет полный внешний вид в цвете и прозрачности (табл. 3.8).

Таблица 3.8

Режим	Описание
0	По умолчанию, белая строка состояния
1	Черная прозрачная строка состояния
2	Полупрозрачное изображение строки состояния

Ориентация

Ориентация (orientation) определяет то, где именно отображается строка состояния на экране iPhone. Если iPhone используется в режиме альбомной ориентации, то приложение может переориентировать себя, чтобы соответствовать более широкому экрану. Значение, передаваемое с аргументом положения, представляет угол, под которым будет отображаться экран iPhone (табл. 3.9).

Таблица 3.9

Угол	Описание
0	Строка состояния отображается наверху iPhone
90	Строка состояния отображается в альбомной ориентации вдоль правой части экрана
-90	Строка состояния отображается в альбомной ориентации вдоль левой части экрана

Длительность

Длительность (*duration*) определяет число в секундах, которое должен занимать анимационный переход между предыдущим статусом строки состояния и новым. Для мгновенного перехода используйте значение 0.

Fence ID

Этот параметр предназначен для внутренних нужд. Мы даже не догадываемся, что он означает.

Анимация

Анимация (*animation*) задает анимацию для осуществления перехода от предыдущего статуса строки состояния к новому. На сегодня поддерживаются следующие анимации (табл. 3.10).

Таблица 3.10

Анимация	Описание
0	Постепенное исчезновение/постепенное проявление
1	Новая строка состояния появляется на экране снизу
2	Старая строка состояния исчезает с экрана вниз
3	Старая строка состояния исчезает с экрана вверх; новая строка состояния появляется на экране сверху
4	Новая строка состояния появляется на экране снизу с использованием различных анимаций
5	Без анимации

Скрытие строки состояния

Если вы из каких-либо соображений хотите удалить строку состояния, то ее размер должен измениться, чтобы освободить пространство экрана для других видов, и она должна быть скрыта. Размер строки состояния можно установить с помощью класса `UIHardware`.

Чтобы полностью удалить строку состояния, установите ее высоту в 0 и пометьте ее как скрытую:

```
[ UIHardware _setStatusBarHeight: 0.0 ];  
[ self setStatusBarHidden: YES ];
```

Чтобы восстановить строку состояния:

```
[ UIHardware _setStatusBarHeight: 20.0 ];  
[ self setStatusBarHidden: NO ];
```

Изображения строки состояния

На строку состояния можно помещать изображения, чтобы уведомить пользователя о каком-либо конкретном состоянии приложения, например, воспроизведении музыки или о входе в чат (chat room). Изображения остаются на строке состояния даже тогда, когда вы выйдете из вашего приложения, тем самым позволяя пользователю быть в курсе будущих событий, например, будильника. Если ваше приложение завершится раньше времени, то изображение может быть автоматически удалено.

Изображения строки состояния управляются приложением SpringBoard. Это несколько затрудняет настройку, при которой ваше приложение должно скопировать используемые вами изображения в программную папку SpringBoard (и перезапустить SpringBoard) прежде, чем все это заработает.

Чтобы корректно отображать изображения строки состояния, необходимы два файла: один для белой строки состояния, а другой — для черной. В зависимости от того, какое приложение выполняется, используемая строка состояния будет автоматически переключаться к соответствующему изображению.

Установка

При первом запуске вашего приложения оно должно скопировать два файла в папку SpringBoard, расположенную по адресу /System/Library/CoreServices/SpringBoard.app. Для своих изображений выбирайте односложный дескриптор и называйте их Default_NAME.png и FSO_NAME.png. Изображение Default будет отображаться при использовании белой строки состояния, а изображение FSO будет выводиться на черной строке состояния.

После того как эти файлы будут скопированы в папку SpringBoard, чтобы они были распознаны, программу SpringBoard необходимо перезапустить. Для этого вызовите `launchctl` в оболочке iPhone:

```
# launchctl stop com.apple.SpringBoard
```

Либо вы можете поручить пользователю перезагрузить iPhone. Если SpringBoard не перезапустить, то ваша строка состояния будет игнорироваться до тех пор, пока пользователь не перезагрузит свой iPhone.

Отображение и удаление изображения строки состояния

После того как вы все настроите, для отображения новой строки состояния воспользуйтесь методом `addStatusBarImageNamed`, используя придуманный вами ранее однословный дескриптор:

```
[ UIApp addStatusBarImageNamed: @"NAME" removeOnAbnormalExit: YES ];
```

Аргумент `removeOnAbnormalExit` указывает iPhone, нужно ли автоматически удалять изображение в случае краха приложения.

Если вы собираетесь удалить изображение вручную, воспользуйтесь методом `removeStatusBarImageNamed`:

```
[ UIApp removeStatusBarImageNamed: @"NAME" ];
```

Бейджи приложения

Имея все эти многочисленные подключения — EDGE, WiFi и Bluetooth (не говоря уже о сотовой сети), может произойти многое, пока это небольшое устройство будет лежать в вашем кармане. Без каких-либо уведомлений о наличии незавершенных задач пользователь легко может пропустить что-нибудь важное, пока будет погружен в реальную жизнь. Бейджи приложений (*application badges*) — это небольшие кружки, отображаемые на значке программы в SpringBoard. Бейджи приложений активно используются поставляемыми Apple приложениями для уведомления пользователя о пропущенных звонках, полученных голосовых или текстовых сообщениях и сообщениях электронной почты.

Эти бейджи имеют одну очень удобную особенность, состоящую в том, что для отображения бейджей на SpringBoard само приложение необязательно должно быть запущено. Это очень полезно при использовании их в качестве

напоминаний пользователю даже после того, как он выйдет из приложения. Это также означает то, что вам необходимо очистить все старые бейджи при выходе из программы.

Отображение бейджа приложения

Бейджи приложения являются одной из самых простых возможностей воспользоваться всеми предоставляемыми ими преимуществами путем только одного вызова класса `UIApplication`:

```
[ UIApplication setApplicationBadge: @"Hi!" ];
```

Метод `setApplicationBadge` берет объект `NSString`, который может быть построен путем стандартного форматирования строки:

```
NSString *badgeText = [ [ NSString alloc ]  
initWithFormat:@"%d", numNewMessages ];  
[ UIApplication setApplicationBadge: badgeText ];
```

Удаление бейджа приложения

Бейдж приложения должен удаляться, когда пользователь нажимает на страницу с важными событиями, о которых его уведомляет этот бейдж. Удаление бейджа приложения также является весьма простой задачей. Такой код лучше всего поместить после реализации перехода к виду с этими событиями. Например:

```
[ transitionView transition: 0 toView: missedCalls ];  
[ UIApplication removeApplicationBadge ];
```

Бейдж приложения будет отображаться даже после того, как приложение завершит свою работу. Это может быть полезно, но это не всегда то, что вы хотите. Если бейдж приложения должен быть удален при выходе из программы, то вам нужно вызвать `removeApplicationBadge` в методе `applicationWillTerminate` приложения:

```
- (void)applicationWillTerminate {  
    /* Мы собираемся выйти из программы,  
       поэтому удалите бейдж приложения. */  
    [ UIApplication removeApplicationBadge ];  
}
```

В следующем разделе вы узнаете больше об этом типе элементов управления состоянием приложения.

Для дальнейшего изучения

Прежде чем продолжить изучение ситуаций, при которых бейджи приложения могут помочь улучшить реакцию вашего приложения на изменения состояния, проведите некоторые исследования.

- Поэкспериментируйте и определите максимальное количество текста, который может быть добавлен на бейдж приложения. Что происходит, когда вы превышаете этот лимит?
- Проверьте наличие прототипов `UIApplication.h` в каталоге вашего пакета инструментов. Вы найдете их в папке `/toolchain/sys/usr/include/UIKit/`. В *главе 7* мы опишем больше методов `UIApplication`.

Сервисы приложения

Состояние приложения имеет большее значение для iPhone, нежели для настольного компьютера. Это связано с тем, что многие события на iPhone могут приводить к зависанию приложения, фоновой работе приложения или аварийному его завершению. Эти различные состояния возникают, когда пользователь нажимает домашнюю кнопку (home button), блокирует экран или получает входящий звонок. Приложению важно знать об изменении своего состояния, чтобы сохранить все настройки, остановить потоки или выполнить другие действия.

Базовый класс `UIApplication` состоит из многих функций для изменения состояний приложения, которые могут быть подменены приложением. Пока имеется не особо много действий, которые приложение может выполнить в связи с изменением своего состояния, но оно может, по крайней мере, предпринять какие-либо шаги для подготовки к этому.

Приостановка

При нажатии домашней кнопки (home button) по умолчанию приложение приостанавливает свою работу. То же самое происходит во время телефонно-

го разговора или блокировки экрана. При этом вызываются методы приостановки.

В зависимости от природы события может вызываться один из трех методов приостановки.

- `applicationWillSuspendUnderLock` — экран iPhone блокируется либо путем нажатия кнопки выключения, либо во время ожидания блокировки экрана;
- `applicationWillSuspendForEventsOnly` — к вызову этого метода приводят события, которые переводят приложения в фоновый режим, например, получение входящего звонка. Также этот метод вызывается в случаях, когда экран заблокирован, но метод `applicationWillSuspendUnderLock` не подменен;
- `applicationSuspend` — этот метод вызывается, когда пользователь нажимает домашнюю кнопку (home button). Для приложения это последний шанс произвести все необходимые действия. Если ни один из двух других методов не подменен, то вызывается данный метод для приостановки всех событий.

Поскольку один метод закрывает недостатки других, то большинство приложений может обрабатывать все приостановленные события, просто подменив `applicationSuspend`:

```
- (void)applicationSuspend:(struct _GSEvent *)event {  
    /* Мы собираемся приостановить работу,  
       поэтому сделайте здесь что-нибудь. */  
}
```

Поскольку объем памяти и время жизни батареи весьма ограничены, то вопрос ресурсов на iPhone стоит очень остро. Если приложению не нужно выполняться в фоновом режиме, и нет причины удерживать состояние приложения, то имеет смысл просто прекратить работу вместо приостановки. iPhone достаточно умен, чтобы перезапускать последнее использованное приложение, поэтому пользователь может даже не почувствовать разницы:

```
- (void)applicationSuspend:(struct _GSEvent *)event {  
    [ self terminate ];  
}
```

Можно прекращать или приостанавливать работу приложения в зависимости от его состояния. Например, если приложение является клиентом от-

правки мгновенных сообщений, то приостанавливать его работу имеет смысл только, если он имеет действующее подключение к серверу, так что оно будет продолжать свою работу в фоновом режиме и поддерживать подключение. Если пользователь выходит из программы, то нет никакой причины оставлять ее работать, поэтому более оправданным является прекращение ее работы:

```
- (void)applicationSuspend:(struct _GSEvent *)event {
    if (connected == NO) {
        [ self terminate ];
    }else {
        /* Фоновый режим IM Client */
    }
}
```

Когда приложение приостановлено, то оно просто перемещается в фон и может продолжать свое выполнение. Если выполняется отдельный поток для новых событий, например, новых мгновенных сообщений, то оно все еще способно коммуницировать с процессом и даже отправлять звуковые эффекты для предупреждения пользователя о каких-либо событиях (см. главу 6).

Возобновление

Когда приложение возвращается в состояние работы, то для возвращения приложения к жизни вызывается другой набор методов. Они могут быть изменены для проверки (и переустановки) подключения или выполнения других задач.

Аналогично трем методам, использованным ранее для обработки различных типов приостановки, имеются три метода для возобновления:

- `applicationDidResumeFromUnderLock` — приложение, возобновляемое на iPhone, экран которого был заблокирован и выключен;
- `applicationDidResumeForEventsOnly` — возобновление приложения по окончании телефонного разговора или другого события, приведшего к фоновой работе приложения;
- `applicationDidResume` — перехватчик всех остальных событий возобновления.

Методы возобновления подменяются так же, как и их собратья приостановки:

```
- (void)applicationDidResume {  
    /* Мы возобновляемся. Сделайте что-нибудь. */  
}
```

Прекращение работы программы

Если только приложение само не прекратило свою работу, то, как правило, оно не прекращает свою работу, пока не будет завершена работа самого iPhone. Поскольку большинство хороших приложений сами прекращают свою работу (вместо приостановки), то разумно подменить методы прекращения до прекращения работы приложения.

Метод `applicationWillTerminate` вызывается всякий раз, когда приложение собирается аккуратно завершить свою работу. Он *не* вызывается, если приложение убивается путем удерживания домашней кнопки. Этот метод должен осуществить все оставшееся освобождение ресурсов, убедиться в том, что все подключения корректно закрыты, и выполнить все остальные задачи, необходимые для выхода из программы:

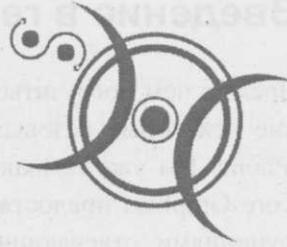
```
- (void)applicationWillTerminate {  
    /* Мы собираемся выйти, поэтому сделайте что-нибудь. */  
}
```

Для дальнейшего изучения

Вот некоторые приемы, которые можно использовать для экспериментов с сервисами приложения.

- Используйте `pthread` или `NSThread` для создания фонового потока, поддерживающего активное подключение к серверу. Что произойдет с подключением, если приложение приостановит свою работу?
- Проверьте наличие прототипов `UIApplication.h` в каталоге вашего пакета инструментов. Вы найдете их в папке `/toolchain/sys/usr/include/UIKit/`. В главе 7 мы опишем больше методов `UIApplication`.

ГЛАВА 4



Обработка событий и платформа Graphics Services

В главе 3 были рассмотрены основные элементы пользовательского интерфейса iPhone. Многие из объектов поддерживают события высокого уровня, такие как `buttonClicked` и `tableViewRowSelected`, уведомляющие приложение о действиях, производимых пользователем. Эти действия полагаются на события мыши более низкого уровня, предоставляемые классом `UIView` и его базовым классом `UIResponder`. Класс `UIResponder` предоставляет методы распознавания и управления основными событиями мыши, которые происходят, когда пользователь касается или проводит по экрану iPhone. Эти методы включены в другие события, созданные в `UIView` для распознавания касаний двумя пальцами. Высокоуровневые объекты, например, таблицы и листы предупреждений, берут эти события низкого уровня и оборачивают их в высокоуровневые, чтобы управлять нажатиями кнопок, выбором строки и другими действиями. Все эти события, относящиеся к экрану, обрабатываются с использованием платформы `Graphics Services`, которая предоставляет координаты экрана, информацию о касаниях пальцами и другие данные, относящиеся к графике данного события.

Эта платформа считалась низкоуровневой и поэтому позднее надстраивалась более дружественным API касаний и другими платформами. Эти функции низкого уровня сообщают приложению, что именно произошло на экране, и предоставляют информацию, необходимую для взаимодействия с пользователем. Они широко распространены в поставляемых Apple приложениях. Для получения многих из этих событий вам потребуется добавить в ваше приложение следующую функцию вызова:

```
UIApplicationUseLegacyEvents (YES);
```

Введение в геометрические структуры

Прежде чем погрузиться в управление событиями, вам потребуется понимание некоторых базовых геометрических структур, обычно используемых в iPhone. Вы уже познакомились с некоторыми из них в *главе 3*. Платформа Core Graphics предоставляет множество основных структур для управления функциями, отвечающих за работу с графикой. К таким структурам относятся точки, размеры окон и области окна. Core Graphics также предоставляет множество функций языка C для создания и сравнения этих структур.

CGPoint

CGPoint — самая простая структура Core Graphics. Она содержит два значения с плавающей запятой, которые соответствуют горизонтальной (*X*) и вертикальной (*Y*) координатам на дисплее. Для создания CGPoint используется метод CGPointMake:

```
CGPoint point = CGPointMake (320.0, 480.0);
```

Первое значение соответствует *X*-координате по горизонтали, и второе — *Y*-координате по вертикали. Эти значения так же могут быть получены напрямую:

```
float x = point.x;
```

```
float y = point.y;
```

Дисплей iPhone имеет разрешение 320×480 точек. Левый верхний угол экрана принимается за начало отсчета с координатами (0, 0), а максимальное значение достигается в правом нижнем углу экрана (319, 479). Отсчет координат идет от нуля.

Будучи универсальной структурой, CGPoint может одинаково хорошо ссылаться на координату как экрана, так и области окна. Например, если окно имеет размеры 160×240 (половина экрана), то CGPoint со значением (0, 0) может адресовать либо левый верхний угол экрана, либо левый верхний угол окна (0×240). Что конкретно адресует эта структура, определяется из контекста использования структуры в программе. Две структуры CGPoint можно сравнить с помощью функции CGPointEqualToPoint:

```
BOOL isEqual = CGPointEqualToPoint (point1, point2);
```

CGSize

Структура CGSize представляет размер прямоугольника. Она инкапсулирует ширину и высоту объекта и изначально находится в API iPhone, задавая размер объектов экрана, т. е. окон. Чтобы создать объект CGSize, воспользуйтесь CGSizeMake:

```
CGSize size = CGSizeMake(320.0, 480.0);
```

Значения, передаваемые в CGSizeMake, указывают ширину и высоту описываемого элемента. Эти значения можно получить напрямую с помощью имен переменных width и height структуры:

```
float width = size.width;  
float height = size.height;
```

Сравнить две структуры CGSize можно с помощью функции CGSizeEqualToSize:

```
BOOL isEqual = CGSizeEqualToSize(size1, size2);
```

CGRect

Структура CGRect объединяет структуры CGPoint и CGSize для описания области окна на экране. Область окна описывается origin, представляющей местоположение левого верхнего угла окна, и size — размером окна. Для создания CGRect используется функция CGRectMake:

```
CGRect rect = CGRectMake(0, 200, 320, 240);
```

Этот пример описывает окно 320×240, левый верхний угол которого расположен в точке координат (0, 200). Как и в случае со структурой CGPoint, эти координаты могут соответствовать либо самой точке на экране, либо смещению внутри существующего окна. Это зависит от того, где и как используется структура CGRect.

К компонентам структуры CGRect также можно получить доступ напрямую:

```
CGPoint windowOrigin = rect.origin;  
float x = rect.origin.x;  
float y = rect.origin.y;
```

```
CGSize windowSize = rect.size;  
float width = rect.size.width;  
float height = rect.size.height;
```

Включение и пересечение

Две структуры `CGRect` можно сравнить с помощью функции `CGRectEqualToRect`:

```
BOOL isEqual = CGRectEqualToRect(rect1, rect2);
```

Чтобы определить, содержится ли заданная точка в `CGRect`, воспользуйтесь методом `CGRectContainsPoint`. Это особенно полезно при определении того, коснулся ли пользователь экрана в какой-либо конкретной области. Точка представляется в виде структуры `CGPoint`:

```
BOOL containsPoint = CGRectContainsPoint(rect, point);
```

Подобная функция может использоваться для определения, содержит ли одна структура `CGRect` другую структуру `CGRect`. Это полезно при выяснении того, перекрываются ли заданные объекты:

```
BOOL containsRect = CGRectContainsRect(rect1, rect2);
```

Чтобы определить, пересекаются ли две структуры `CGRect`, используйте функцию `CGRectIntersectsRect`:

```
BOOL doesIntersect = CGRectIntersectsRect(rect1, rect2);
```

Обнаружение границы и центра

Следующие функции могут использоваться для определения различных границ прямоугольника и вычисления координат центра прямоугольника. Все эти функции принимают структуру `CGRect` в качестве своего единственного аргумента и возвращают значение типа `float`:

- `CGRectGetMinX` — возвращает координату левой границы прямоугольника;
- `CGRectGetMinY` — возвращает координату нижней границы прямоугольника;
- `CGRectGetMidX` — возвращает X -координату центра прямоугольника;
- `CGRectGetMidY` — возвращает Y -координату центра прямоугольника;
- `CGRectGetMaxX` — возвращает координату правой границы прямоугольника;
- `CGRectGetMaxY` — возвращает координату верхней границы прямоугольника.

Введение в GSEvent

Структура `GSEvent` — стандартный объект, описывающий события графического уровня как методы обработки событий класса. Он может быть использован совместно с платформой Graphics Services для расшифровки детальной информации о произошедшем событии.

При получении уведомления о каком-либо событии все методы события получают указатель на структуру `GSEvent`. Прототип события, как правило, следует такому стандарту:

```
- (void)eventName: (struct _GSEvent *)event
{
    /* Код обработки события */
}
```

Graphics Services

Всякий раз при получении события объект взаимодействует с платформой Graphics Services для получения детальной информации о событии. Платформа Graphics Services предоставляет множество различных функций расшифровки для извлечения информации о событии.

Местоположение события

Для событий одного касания функция `GSEventGetLocationInWindow` возвращает структуру `CGPoint`, содержащую координаты X , Y места, где произошло событие. Эти координаты, как правило, являются смещением относительно положения окна, которое получило событие. Например, если окно находится в нижней половине экрана, с координатами левого верхнего угла $(0, 240)$, и событие получено в точке $(0, 0)$, то это означает, что событие на самом деле произошло на экране в точке $(0, 240)$, в которой расположен левый верхний угол $(0, 0)$ окна.

Метод `GSEventGetLocationInWindow` возвращает структуру `CGPoint`, которую вы можете распаковать следующим образом:

```
CGPoint point = GSEventGetLocationInWindow(event);
float x = point.x;
float y = point.y;
```

Если используется касание двумя пальцами, то вы должны вызвать две отдельные функции, чтобы получить координаты окна для каждого пальца. Крайнее левое касание считается внутренним, а крайнее правое касание — внешним. Методы `GSEventGetInnerMostPathPosition` и `GSEventGetOuterMostPathPosition` возвращают структуры `CGPoint`, содержащие координаты X и Y окна для каждого касания:

```
CGPoint leftFinger = GSEventGetInnerMostPathPosition(event);
CGPoint rightFinger = GSEventGetOuterMostPathPosition(event);
```

Когда ориентация iPhone является альбомной, эти позиции меняются местами:

```
int orientation = [ UIHardware deviceOrientation: YES ];

if (orientation == kOrientationHorizontalLeft ||
    orientation == kOrientationHorizontalRight)
{
    leftFinger = GSEventGetOuterMostPathPosition(event);
    rightFinger = GSEventGetInnerMostPathPosition(Event);
}
```

Тип события

Тип события определяет, использовалось ли касание одним пальцем, скользил ли палец по экрану или был убран с экрана.

Большинство событий может быть легко определено посредством метода, который получил уведомление. Например, если палец коснулся экрана, то уведомляется метод `mouseDown`, тогда как в результате касания двумя пальцами уведомляется метод `gestureStarted`:

```
unsigned int eventType = GSEventGetType(event);
```

Для события касания одним пальцем событие началось бы с одного опущенного пальца, а продолжилось бы всеми поднятыми пальцами (когда пользователь уберет палец с экрана). Касание двумя пальцами несколько сложнее. Оно начинается с одного опущенного пальца, и затем переходит в касание двумя пальцами. Если пользователь поднимает один палец, то тип события меняется на событие "один палец поднят", за ним следует событие "все пальцы подняты", когда пользователь убирает и второй палец.

События связаны со значениями типа событий так, как указано в табл. 4.1.

Таблица 4.1

Тип	Описание
1	Один палец опущен, включая первый палец в касании
2	Все пальцы подняты
5	Один палец поднят при касании двумя пальцами
6	Касание двумя пальцами

Событие аккорда (события многократного касания)

Если на фортепьяно сыграть более чем одну ноту, то это будет аккорд. Такая же логика используется при обработке касаний. Прикосновение к экрану одним пальцем представляет собой одну ноту, тогда как два касания рассматриваются как аккорд.

Для определения количества пальцев, коснувшихся экрана в момент возникновения события, может использоваться метод `GSEventIsChordingHandEvent`:

```
int eventChording = GSEventIsChordingHandEvent(event);
```

Эта функция возвращает значение 0, если произошедшее событие — событие касания одним пальцем, и значение 1, если это событие считается аккордом.

События мыши

Событиями мыши считаются любые события касания экрана одним пальцем. Все классы, производные от класса `UIResponder`, наследуют события мыши, а некоторые классы подменяют их, чтобы сформировать новые типы событий, такие как выбор строк в таблице или изменение положения переключателей или бегунков в элементе управления.

Чтобы получить уведомления для любого из шести поддерживаемых событий мыши, создайте подкласс объекта, для которого вы хотите получить события, и подмените его методы. Например, приведенный далее класс получит события, отправленные объекту `UITable`, и определит собственную обработку пользовательских действий, производимых одним пальцем. Это

приведет к тому, что всякий раз, когда пользователь будет нажимать пальцем в пределах области окна с таблицей, будут вызываться данные методы:

```
@interface MyTable : UITable
{
}
- (void) mouseDown:(struct _GSEvent *)event;
- (void) mouseUp:(struct _GSEvent *)event;
@end
```

Поскольку базовый класс может также воспользоваться событием мыши, то вы захотите вызвать версию суперкласса метода либо до, либо после того, как обработаете событие:

```
- (void) mouseDown:(struct _GSEvent *)event {
    /* Обработка отпущания мыши */
    [ super mouseDown: event ];
}
```

mouseDown

Метод `mouseDown` вызывается всякий раз, когда вы нажимаете одним пальцем на экран, включая и первое нажатие пальцем при двойном касании. Местоположение события представляет координаты, в которых произошло нажатие экрана в пределах окна объекта:

```
- (void) mouseDown:(struct _GSEvent *)event {
    CGPoint pointDown = GSEventGetLocationInWindow(event);

    /* Обработка отпущания мыши */

    [ super mouseDown: event ];
}
```

mouseUp

Метод `mouseUp` события получает уведомление всякий раз, когда пользователь убирает палец с экрана. Этот метод больше всех подходит для того, чтобы выполнять проверку элементов управления, таких как сегментированный элемент управления, или других классов, которые не имеют собственных

уведомлений для таких событий. Первым должен вызываться метод супер-класса, чтобы значения элементов управления успели измениться прежде, чем они будут считаны. Местоположение события представляет последние координаты пальца пользователя, перед тем как он был убран с экрана:

```
- (void) mouseUp:(struct _GSEvent *)event {  
    CGPoint pointUp = GSEventGetLocationInWindow(event);  
  
    [ super mouseUp: event ];  
  
    /* Проверка значения элемента управления и т. п. */  
}
```

mouseDragged

Метод `mouseDragged` получает уведомление, если пользователь продолжает удерживать палец после отправки события `mouseDown`, и если палец перемещается со своего исходного положения на экране. Этот метод является аналогом функции перетаскивания на рабочем столе. Местоположение события представляет собой координаты точки, куда пользователь переместил свой палец прежде, чем убрать его с экрана:

```
- (void) mouseDragged:(struct _GSEvent *)event {  
    CGPoint movedTo = GSEventGetLocationInWindow(event);  
  
    [ super mouseDragged: event ];  
  
    /* Обработка перетаскивания мыши */  
}
```

Этот метод следует за пальцем пользователя, поэтому он вызывается через определенные промежутки времени по мере перемещения пальца.

mouseEntered, mouseExited, mouseMoved

На рабочем столе эти методы уведомляют приложение о перемещениях указателя мыши внутри, за пределами или в пределах фрейма объекта при отсутствии щелчка кнопкой мыши. Поскольку в iPhone как таковой мыши нет, то ваше касание пальцем экрана рассматривается как щелчок мышью. Эти

методы бесполезны для iPhone, но последующие мобильные устройства Apple, возможно, будут использовать мышь или трекбол.

События жестов

Когда пользователь переключается с касания одним пальцем на касание двумя пальцами, то это рассматривается как начало жеста (*gesture*). Это приводит к созданию событий жеста, которые могут быть перехвачены путем подмены соответствующих методов. Жесты представлены в базовом классе `UIView`, и только унаследовавшие от него объекты поддерживают их.

Порядок событий следующий: в момент начала жеста вызывается метод `gestureStarted`. Затем, если пользователь меняет положение своего пальца, вызывается метод `gestureChanged` для уведомления объекта о каждом новом положении жеста. При завершении жеста пользователем вызывается метод `gestureEnded`.

Чтобы отправить события жеста, класс должен подменить метод `canHandleGestures`, который возвращает значение типа `Boolean`. Возвращая значение `YES`, вы приказываете iPhone отправить события:

```
- (BOOL) canHandleGestures
{
    return YES;
}
```

gestureStarted

Метод `gestureStarted` получает уведомление, когда пользователь дотрагивается до экрана двумя пальцами или переходит от использования одного пальца к использованию двух. Этот метод является версией метода `mouseDown` для двух пальцев. Внутренние и внешние координаты соответствуют первой точке прикосновения к экрану. Возвращаемые положения события в структурах `CGPoint` представляют координаты точек, в которых произошли касания каждым пальцем:

```
- (void) gestureStarted: (struct _GSEvent) event {
    CGPoint leftFinger = GSEventGetInnerMostPathPosition(event);
    CGPoint rightFinger = GSEventGetOuterMostPathPosition(event);
}
```

```
[ super gestureStarted: event ];  
  
/* Обработка события начала касания */  
}
```

gestureEnded

Метод `gestureEnded` является аналогом метода `mouseUp` для двух пальцев и сообщает приложению, что пользователь убрал с экрана, по крайней мере, один палец. Если пользователь одновременно убирает оба пальца, то iPhone отправляет оба события методам `gestureEnded` и `mouseUp`. Если пользователь убирает пальцы поочередно, то поднятие первого пальца приводит к вызову метода `gestureEnded`, а второго — к вызову метода `mouseUp`.

Координаты экрана, предоставляемые методом `gestureEnded`, определяют точку, в которой палец остается прижатым к экрану, в то время как другой палец уже убран. Когда оставшийся прижатым палец будет убран с экрана, будет уведомлен метод `mouseUp`, поскольку как только убирается один палец, то жест становится событием мыши:

```
- (void)gestureEnded:(struct _GSEvent)event {  
    CGPoint leftFinger = GSEventGetInnerMostPathPosition(event);  
    CGPoint rightFinger = GSEventGetOuterMostPathPosition(event);  
  
    [ super gestureEnded: event ];  
  
    /* Обработка завершения жеста */  
}
```

gestureChanged

Метод `gestureChanged` вызывается всякий раз, когда пользователь перемещает свои пальцы во время жеста. Этот метод является аналогом метода `mouseDragged` для двух пальцев. Когда это происходит, приложение должно переоценить положения пальцев в жесте и отреагировать соответствующим образом. Местоположения события представляют новые координаты, куда были передвинуты пальцы:

```
- (void)gestureChanged:(struct _GSEvent)event {  
    CGPoint leftFinger = GSEventGetInnerMostPathPosition(event);  
    CGPoint rightFinger = GSEventGetOuterMostPathPosition(event);
```

```
[ super gestureChanged: event ];  
  
/* Обработка события изменения жеста */  
}
```

События строки текущего состояния

Строка состояния iPhone сама по себе является окном, способным получать события мыши. Приложение Safari создает прецедент действий, которые должны выполняться при таких событиях, т. е. когда основной вид прокручивается к началу. Класс `UIApplication` содержит три уведомления строки состояния, которые могут быть подменены в вашей программе.

Событие `statusBarMouseDown` получает уведомление всякий раз, когда пользователь касается строки состояния:

```
- (void)statusBarMouseDown:(struct _GSEvent *)event;
```

Если пользователь опускает палец и перемещает его в другое место, то метод `statusBarMouseDragged` получает уведомление с событием, содержащим координаты места на экране, к которому его перетаскивали:

```
- (void)statusBarMouseDragged:(struct _GSEvent *)event;
```

Наконец, когда пользователь поднимает палец, метод `statusBarMouseUp` получает уведомление с событием, содержащим координаты точки отрыва пальца пользователя от экрана:

```
(void)statusBarMouseUp:(struct _GSEvent *)event;
```

Пример: перетаскивание значка

Этот пример создает на экране четыре значка и позволяет пользователю свободно их перемещать либо каждый по отдельности, либо по два сразу с помощью жеста. Тем самым иллюстрируется использование различных геометрических структур и функций, уведомлений событий и функций платформы `Graphics Services`. Чтобы подцепить значок, коснитесь его пальцем и не отпускайте, переместите, куда требуется, а затем отпустите палец. Если коснуться строки состояния, то значки вернуться на свои исходные позиции.

Чтобы скомпилировать этот пример из командной строки, помимо основных платформ вы должны будете использовать несколько других платформ: Core Graphics, Graphics, Services и UIKit:

```
$ arm-apple-darwin9-gcc -o MyExample MyExample.m -lobjc \
    -framework CoreFoundation -framework Foundation \
    -framework UIKit -framework CoreGraphics \
    -framework GraphicsServices
```

Листинги 4.1 и 4.2 содержат файлы заголовков и исполняемые методы примера.

Листинг 4.1. Пример работы с мышью и жестом (MyExample.h)

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

@interface MainView : UIView
{
    UIImage *images[4];
    CGRect positions[4];
    CGPoint offsets[4];
    int dragLeft, dragRight;
}
- (id)initWithFrame:(struct CGRect)windowRect;
- (void)reInit;
- (void)mouseDown: (struct _GSEvent *)event;
- (void)mouseUp: (struct _GSEvent *)event;
- (void)mouseDragged: (struct _GSEvent *)event;
- (void)gestureStarted: (struct _GSEvent *)event;
- (void)gestureEnded: (struct _GSEvent *)event;
- (void)gestureChanged: (struct _GSEvent *)event;
- (void)drawRect:(CGRect)rect;
@end

@interface MyApp : UIApplication
{
```

```
    UIWindow *window;
    MainView *mainView;
}
- (void)applicationDidFinishLaunching:(NSNotification *)aNotification;
- (void)statusBarMouseDown:(struct _GSEvent *)event;
@end
```

Листинг 4.2. Пример работы с мышью и жестом (MyExample.m)

```
#import <Foundation/Foundation.h>
#import <CoreFoundation/CoreFoundation.h>
#import <GraphicsServices/GraphicsServices.h>
#import "MyExample.h"

int main(int argc, char **argv)
{
    NSAutoreleasePool *autoreleasePool = [
        [NSAutoreleasePool alloc] init
    ];
    UIApplicationUseLegacyEvents(YES);
    int returnCode = UIApplicationMain(argc, argv, @"MyApp", @"MyApp");
    [autoreleasePool release];
    return returnCode;
}

@implementation MyApp
- (void)applicationDidFinishLaunching:(NSNotification *)aNotification {
    UIWindow *window = [ [UIWindow alloc] initWithContentRect:
        [UIHardware fullScreenApplicationContentRect]
    ];

    CGRect rect = [UIHardware fullScreenApplicationContentRect];
    rect.origin.x = rect.origin.y = 0.0f;

    MainView *mainView = [ [MainView alloc] initWithFrame: rect ];
    [window setContentView: mainView];
}
```

```
[ window orderFront: self ];
[ window makeKey: self ];
[ window _setHidden: NO ];
}

- (void)statusBarMouseDown:(struct _GSEvent *)event {
    [ mainWindow reInit ];
    [ mainWindow setNeedsDisplay ];
}

@end

@implementation MainView
- (id)initWithFrame:(struct CGRect)windowRect {
    self = [ super initWithFrame: windowRect ];
    if (nil != self) {
        int i;

        images[0] = [ UIImage
            imageAtPath: @"/Applications/MobilePhone.app/icon.png" ];
        images[1] = [ UIImage
            imageAtPath: @"/Applications/MobileMail.app/icon.png" ];
        images[2] = [ UIImage
            imageAtPath: @"/Applications/MobileSafari.app/icon.png" ];
        images[3] = [ UIImage
            imageAtPath: @"/Applications/MobileMusicPlayer.app/icon.png" ];

        [ self reInit ];
    }
    return self;
}
- (void)reInit {
    positions[0] = CGRectMake(98, 178, 60, 60);
    positions[1] = CGRectMake(162, 178, 60, 60);
    positions[2] = CGRectMake(98, 242, 60, 60);
    positions[3] = CGRectMake(162, 242, 60, 60);
}
```

```
        dragLeft = dragRight = -1;
    }
- (void)drawRect:(CGRect)rect {
    float black[4] = { 0, 0, 0, 1 };
    CGContextRef ctx = UIGraphicsGetCurrentContext();
    int i;

    CGContextSetFillColor(ctx, black);
    CGContextFillRect(ctx, rect);

    for(i=0;i<4;i++) {
        [ images[i] drawPartImageInRect: positions[i] ];
    }
}

- (void)mouseDown: (struct _GSEvent *)event {
    CGPoint point = GSEventGetLocationInWindow(event);
    int i;

    for(i=0;i<4;i++) {
        if (CGRectContainsPoint(positions[i], point)) {
            dragLeft = i;
            offsets[i] = CGPointMake(point.x - positions[i].origin.x,
                                     point.y - positions[i].origin.y);
        }
    }
}

- (void)mouseUp: (struct _GSEvent *)event {
    CGPoint point = GSEventGetLocationInWindow(event);
    int i;

    dragLeft = -1;
}
}
```

```
- (void)mouseDragged: (struct _GSEvent *)event {
    CGPoint point = GSEventGetLocationInWindow(event);
    CGRect old;
    int i;

    if (dragLeft != -1) {
        old = positions[dragLeft];
        positions[dragLeft].origin.x = point.x - offsets[dragLeft].x;
        positions[dragLeft].origin.y = point.y - offsets[dragLeft].y;
        [ self setNeedsDisplayInRect: old ];
        [ self setNeedsDisplayInRect: positions[dragLeft] ];
    }
}

- (void)gestureStarted: (struct _GSEvent *)event {
    CGPoint leftFinger = GSEventGetInnerMostPathPosition(event);
    CGPoint rightFinger = GSEventGetOuterMostPathPosition(event);
    int i;

    for (i=0;i<4;i++) {
        if (CGRectContainsPoint(positions[i], leftFinger)) {
            dragLeft = i;
            offsets[i] = CGPointMake(leftFinger.x - positions[i].origin.x,
                                     leftFinger.y - positions[i].origin.y);
        }
        else if (CGRectContainsPoint(positions[i], rightFinger)) {
            dragRight = i;
            offsets[i] = CGPointMake(rightFinger.x - positions[i].origin.x,
                                     rightFinger.y - positions[i].origin.y);
        }
    }
}

- (void)gestureEnded: (struct _GSEvent *)event {
    CGPoint leftFinger = GSEventGetInnerMostPathPosition(event);
```

```
CGPoint rightFinger = GSEventGetOuterMostPathPosition(event);
int i;

dragLeft = dragRight = -1;

for (i=0;i<4;i++) {
    if (CGRectContainsPoint(positions[i], leftFinger))
        dragLeft = i;
    else if (CGRectContainsPoint(positions[i], rightFinger))
        dragRight = i;
}
}

- (void)gestureChanged: (struct _GSEvent *)event {
    CGPoint leftFinger = GSEventGetInnerMostPathPosition(event);
    CGPoint rightFinger = GSEventGetOuterMostPathPosition(event);
    CGRect old;
    int i;

    if (dragLeft != -1) {
        old = positions[dragLeft];
        positions[dragLeft].origin.x=leftFinger.x - offsets[dragLeft].x;
        positions[dragLeft].origin.y=leftFinger.y - offsets[dragLeft].y;
        [ self setNeedsDisplayInRect: old ];
        [ self setNeedsDisplayInRect: positions[dragLeft] ];
    }

    if (dragRight != -1) {
        old = positions[dragRight];
        positions[dragRight].origin.x
            = rightFinger.x - offsets[dragRight].x;
        positions[dragRight].origin.y
            = rightFinger.y - offsets[dragRight].y;
        [ self setNeedsDisplayInRect: old ];
    }
}
```

```
        [ self setNeedsDisplayInRect: positions[dragRight] ];
    }
}

- (BOOL) canHandleGestures {
    return YES;
}

@end
```

Как это работает

Вот как работает перемещение значков:

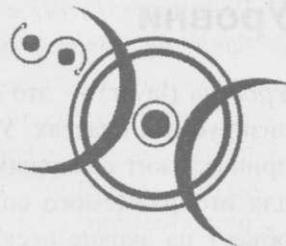
1. При порождении приложения создается объект `MainView`, являющийся производным от `UIView`, и вызывается его метод `initWithFrame`. При этом инициализируются изображения и положения на экране четырех значков: **Phone**, **Mail**, **Safari** и **iPod**. Затем классу вида приказывается отобразить себя.
2. Когда класс вида отобразится, вызывается метод `drawRect` этого класса. Это приводит к прорисовке на пустом поле экрана черного прямоугольника. Затем на экране по отдельности прорисовывается каждый значок с помощью методов класса `UIImage` (более подробно обсуждается в главе 7).
3. Когда для перемещения значка используется один палец, то первым вызывается метод `mouseDown`. Он проверяет, на какой значок нажал пользователь, и устанавливает его в переменной `dragLeft` объекта как перемещаемый значок. Дельта между тем, где пользователь нажал на значок, и левым верхним (исходным) углом значка, сохраняется, чтобы пример мог четко отследить ту часть значка, которую пользователь нажал пальцем.
4. Когда палец пользователя перемещается, вызывается метод `mouseDragged`, который устанавливает положение значка в текущее положение пальца в соответствии с тем, в каком именно месте пользователь нажал на значок. Таким образом, если пользователь нажал в центр значка, значок перемещается так, что его центр следует за пальцем пользователя. Метод `setNeedsDisplayInRect` вызывается, чтобы снова вызвать метод класса вида `drawRect`.
5. Когда пользователь поднимает палец, вызывается метод `mouseUp`, который сбрасывает активный значок.

6. Если используются два пальца, методы жеста выполняют те же самые задачи, но обрабатывают положения обоих пальцев, позволяя тем самым двум значкам следовать за пальцами пользователя. Поскольку вы не знаете, будут ли пальцы опущены и подняты в том же самом порядке, то информация для левого и правого пальцев должна храниться отдельно. Обратите внимание, что опускание второго пальца заставляет отбросить информацию, сохраненную методом `mouseDown`, и заменить ее информацией, возвращаемой методом `gestureStarted`. Аналогично поднятие второго пальца заставляет отбросить информацию, сохраненную методом `gestureChanged`, и заменить ее информацией, возвращаемой методом `mouseDragged`.

Для дальнейшего изучения

Было бы неплохо изучить все различные методы, поддерживаемые в этих низкоуровневых классах. Проверьте следующие прототипы в папке `include` вашего пакета инструментов. Они могут находиться в папке `/toolchain/sys/usr/include: UIKit/UIResponder.h, UIKit/UIView.h, GraphicsServices/GraphicsServices.h` и `CoreGraphics/CGGeometry.h`.

ГЛАВА 5



Графическое программирование с использованием Core Surface и Quartz Core для опытных пользователей

Core Surface — это платформа на базе C, используемая для построения видеобuffers, в которых необработанные пиксельные данные могут писаться напрямую в уровень экрана. Эта платформа используется, когда необходима более совершенная визуализация, нежели простые графические файлы, и предназначена для приложений, использующих собственную графику, создаваемую в процессе работы приложения. Поддерживается множество различных форматов пикселей, а поскольку поверхности имеют прямой интерфейс к уровню экрана, то прорисовка происходит гораздо быстрее, чем при использовании классов изображений более высокого уровня.

Платформа Quartz Core, ранее известная как Quartz Core, предоставляет базовые классы для управления уровнями в рамках объектов вида. Она также используется для склейки буфера Core Surface с объектом на экране и создания 3D-трансформаций 2D-объектов для анимаций и других графических эффектов.

В этой главе вы познакомитесь с обеими платформами и увидите, как они могут взаимодействовать друг с другом для управления поверхностями экрана и создания различных эффектов.

Уровни

Уровень (layer) — это объект низкого уровня, имеющийся во многих визуализируемых классах. Уровни аналогичны листам доски объявлений, которой принадлежит содержимое объекта. Уровень ведет себя как гибкая подложка для отображаемого содержимого объекта и может сгибаться или скручивать объект на экране несколькими способами. Каждый объект, который может визуализировать себя (а таковыми являются объекты, которые порождены от класса `UIView`), имеет, по крайней мере, один уровень, к которому привязано его содержимое.

Например, класс `UIImageView` содержит всю основную информацию об изображении: область его отображения, разрешение и различные методы для работы с изображением. Само изображение привязано к некоторому уровню, а этот уровень используется для отображения данного изображения. Большинство основных уровней ведет себя как единый чистый лист и всего лишь отображает изображение, как оно есть. Более продвинутые классы, например, `UICompositeImageView` (описываемый в главе 7), состоят из множества уровней, которые рассматриваются как диапозитивы, каждый со своим содержимым, сложенные друг на друга для создания одного составного изображения.

Будучи гибкими, уровни могут использоваться для управления изображением. Класс уровня `CALayer` управляет тем, как ведет себя эта "гибкая подложка" после своего отображения. Если вы согнете уровень, то вместе с ним будет согнуто и изображение. Если вы повернете уровень, то изображение тоже будет повернуто. Можно абсолютно легко и просто настраивать прозрачность уровня (`alpha`), добавлять анимацию или же поворачивать, наклонять и масштабировать объект. Объект, находящийся на вершине уровня, совершенно не помнит о том, как с ним поступали, позволяя вашей программе продолжать видеть изображение (или другой объект), как если бы это был все еще 2D-объект. Однако когда пользователь видит изображение, то он принимает ту форму, которую получил уровень.

Уровень не ограничивается только хранением содержимого изображения. В уровне также располагается вывод экрана для `UITextView` или любого другого класса вида, описанного в главе 3. Это означает, что панели навигации, таблицы, текстовые поля и многие другие типы классов видов могут изменяться, масштабироваться и даже анимироваться.

Относительно уровней важно запомнить то, что все объекты `UIView` имеют не менее одного уровня, и они определяют способ, которым, в конечном счете, содержимое выводится на экран.

Поверхности экрана

Поверхность экрана (`screen surface`) — это объект `Core Surface`, используемый для получения доступа к буферу необработанных пикселей, выводимых напрямую в уровень. Он позволяет использовать высокоэффективную 2D-графику в видеопроигрывателях, современных играх или других приложениях, которым необходимо выводить необработанные пиксели прямо на экран. `Core Surface` поддерживает несколько различных конфигураций значений цветов для размещения различных типов пиксельных данных.

Поверхности экрана прикреплены к уровню, а, в конечном счете, поверхности прикреплены к виду. Поверхность привязана к уровню, а уровень добавляется в класс вида, который выводит его на экран. Это также означает, что на выводимое на поверхность экрана изображение можно воздействовать с помощью методов уровня.

Создание поверхности экрана

Поверхность экрана — это объект, содержащий буфер необработанного видео. Он поддерживает особое разрешение, шаг и формат пикселей. Для создания новой поверхности экрана с помощью класса динамического словаря `NSMutableDictionary` инициализируется буфер. Это необходимо для предоставления информации о желаемом внешнем виде и поведении поверхности. `NSMutableDictionary` — это класс, используемый как на настольных, так и на мобильных платформах `Mac OS X` и являющийся частью платформы `Core Foundation`. Полную документацию для этого класса можно найти на [Web-узле Apple Developer Connection](#).

Приведенный далее пример создает объект словаря, задавая видеобуфер `320×480`. Он использует `RGBA`, четырехбайтовый формат пикселей, содержащий по одному байту для красного, зеленого, голубого каналов и канала альфа-сопряжения (`alpha blending`). Последний вызываемый в примере метод

выделяет достаточное количество места в словаре для всех четырехбайтовых пикселей в прямоугольнике $X \times Y$:

```
CFMutableDictionaryRef dict;
int x = 320, y = 480, pitch = x*4, size = 4*x*y;
char *pixelFormat = "RGBA";

dict = CFDictionaryCreateMutable(kCFAllocatorDefault, 0,
    &kCFTypeDictionaryKeyCallBacks, &kCFTypeDictionaryValueCallBacks);
CFDictionarySetValue(dict, kCoreSurfaceBufferGlobal, kCFBooleanTrue);
CFDictionarySetValue(dict, kCoreSurfaceBufferPitch,
    CFNumberCreate(kCFAllocatorDefault, kCFNumberSInt32Type, &pitch));
CFDictionarySetValue(dict, kCoreSurfaceBufferWidth,
    CFNumberCreate(kCFAllocatorDefault, kCFNumberSInt32Type, &x));
CFDictionarySetValue(dict, kCoreSurfaceBufferHeight,
    CFNumberCreate(kCFAllocatorDefault, kCFNumberSInt32Type, &y));
CFDictionarySetValue(dict, kCoreSurfaceBufferPixelFormat,
    CFNumberCreate(kCFAllocatorDefault, kCFNumberSInt32Type,
    pixelFormat));
CFDictionarySetValue(dict, kCoreSurfaceBufferAllocSize,
    CFNumberCreate(kCFAllocatorDefault, kCFNumberSInt32Type, &size));
```

После того как будет построен словарь, для определения типа требуемого буфера можно создать поверхность экрана с использованием его свойств:

```
CoreSurfaceBufferRef screenSurface = CoreSurfaceBufferCreate(dict);
```

Отображение поверхности экрана

Прежде чем поверхность экрана сможет быть отображена, она должна быть привязана к какому-либо уровню. С помощью объекта Quartz Core `CALayer` создайте новый уровень:

```
CALayer *screenLayer;

screenLayer = [ [ CALayer layer ] retain ];
[ screenLayer setFrame: viewRect ];
[ screenLayer setOpaque: YES ];
```

Теперь прикрепите содержимое поверхности экрана к только что созданному уровню. Заметьте, что в процессе обработки поверхности экрана вы обязаны всегда блокировать ее:

```
CoreSurfaceBufferLock(screenSurface, 3);
[ screenLayer setContents: screenSurface ];
CoreSurfaceBufferUnlock(screenSurface);
```

Объекты `UIView`, с которыми вы работаете на протяжении этой книги, содержат собственные базовые уровни. Для отображения содержимого поверхности добавьте только что созданный уровень к существующему `UIView` в качестве подуровня:

```
[ [ self _layer ] addSublayer: screenLayer ];
```

Теперь поверхность экрана привязана к уровню, а уровень добавлен к виду. Если в объекте `UIView` существуют другие уровни, то новый уровень будет помещен поверх предыдущих уровней. Это означает, что вам придется настроить его прозрачность (`alpha`), чтобы увидеть нижележащие уровни.

Вывод на поверхность экрана

Даже после того как вы добавите уровень поверхности к виду, ничего не будет отображено, поскольку сам видеобuffer пуст. Объект `CoreSurfaceBuffer` содержит указатель на необработанный видеобuffer. Чтобы получить доступ к базовому адресу этого необработанного видеобufferа, воспользуйтесь функцией `CoreSurfaceBufferGetBaseAddress`:

```
unsigned long *baseAddress = CoreSurfaceBufferGetBaseAddress(screenSurface);
```

Буфер является одномерным массивом, даже несмотря на то, что ваше приложение и пользователь рассматривают его как двумерный. Пиксели записываются слева направо и сверху вниз. Например, `baseAddress[0]` адресует левый верхний пиксел на поверхности, `baseAddress[319]` адресует правый верхний пиксел для разрешения 320×480, используемого в данной поверхности, а `baseAddress[320]` адресует самый крайний левый пиксел во второй строке.

Поскольку в данном примере используется тип пикселей `RGBA`, то каждый пиксел имеет длину в четыре байта. Использование указателя `unsigned long` (который также имеет длину в четыре байта) позволяет всем пикселям быть адресованными как элементы в массиве.

Вы также можете предпочесть использование указателя `unsigned char`, который позволит вам адресовать отдельные значения пиксела (один байт для каждого канала). Однако имейте в виду, что для того чтобы получить следующий пиксел, вам необходимо увеличивать ваш указатель на четыре байта за раз (исходя из четырехбайтового типа пикселов). Например, `(unsigned char *) baseAddress[0]` ссылается на красный канал первого пиксела, `baseAddress[1]` — на зеленый канал, `baseAddress[2]` — на голубой канал, `baseAddress[3]` — на альфа-канал и, наконец, `baseAddress[4]` — на красный канал следующего пиксела.

16-битные форматы пикселов

16-битные форматы пикселов широко используются вместо 32-битного формата пикселов RGBA для обеспечения большего быстродействия, когда точность передачи цветов не является критичной, а еще потому, что только дорогие настольные дисплеи могут поддерживать оборудование реалистичной 32-битной цветопередачи, и нет особых причин использовать ее на сотовых телефонах. Поскольку в наличии имеется не так уж много памяти, то 16-битный фрейм может быть скопирован в два раза быстрее 32-битного. Чтобы использовать этот формат пикселов, до создания поверхности экрана внесите следующие изменения в свойства словаря этой поверхности:

```
int x = 320, y = 480, pitch = x*2, size = 2*x*y;
char *pixelFormat = "565L";
```

16-битный формат пикселов для каждого пиксела использует по два байта и обозначается как формат пикселов 565L. После того как поверхность будет создана, доступ к ней можно получить с помощью указателя `unsignedshort`, который имеет длину не четыре, а два байта:

```
unsigned short *baseAddress =
    CoreSurfaceBufferGetBaseAddress(screenSurface);
```

Чтобы конвертировать значения RGB в 16-битные значения, подходящие для записи в такой буфер, вы можете определить макрос, позволяющий вашему приложению плавно перемещаться между 16-битным и 32-битным RGB:

```
#define RGB2565L(R, G, B) ((R >> 3) << 11) | ((G >> 2) << 5)
    | ((B >> 3) << 0)
```

Буфер фрейма

Буфер фрейма (frame buffer) — это вторичный буфер памяти, используемый для построения видеофрейма прежде, чем он будет отображен на поверхности экрана. Это может помочь предотвратить мерцание приложения, выводящего графику по одной строке за раз (пример — эмуляторы), а также может быть использовано для синхронизации программ, которые должны отображаться во фреймах за секунду. Использование буфера фрейма полезно также для объединения нескольких видеоуровней вместе до отображения финального фрейма. Например, если вы пишете игру, которая рисует несколько различных видеоуровней (фон, спрайты и HUD), то каждый отдельный видеоуровень может быть нарисован во внутреннем буфере фрейма до того, как финальный фрейм будет выведен на экран. Настройка и копирование буфера фрейма, если вы решите использовать его, — это работа приложения. Когда поверхность экрана будет создана, ее размер рассчитывается как размер пиксела, умноженный на ширину и высоту поверхности. Чтобы использовать буфер фрейма, создайте буфер такого же размера:

```
workBuffer = malloc(2 * x * y);
```

Теперь переместите всю вашу обработку видео так, чтобы она занимала место в этом рабочем буфере до тех пор, пока программа не закончит вывод фрейма. Когда фрейм будет готов, придет время скопировать его в видеобуфер поверхности. Период между завершением фрейма и выводом следующего фрейма называется периодом v-blank; это такой период, когда рабочий буфер может быть безопасно выведен в буфер поверхности экрана:

```
memcpy(baseAddress, workBuffer, 2 * x * y);
```

```
[ scrollView setNeedsDisplay ];
```

Пример: случайный снег

Первой графической программой, которую пишет любой фанат-программист, является отображение случайных цветов в пределах разрешения экрана. Следуя этой традиции, наш пример создает и отображает поверхность экрана, затем каждые пять секунд применяет `rand()` к каждому пикселу. После запуска примера весь экран будет заполнен сумасшедшим цветным снегом.

Чтобы построить данный пример, необходимо подключить платформы Core Surface и Quartz Core:

```
$ arm-apple-darwin9-gcc -o MyExample MyExample.m -lobjc \
  -framework CoreFoundation -framework Foundation -framework UIKit \
  -framework CoreSurface -framework QuartzCore
```

В листингах 5.1 и 5.2 приведен код приложения.

Листинг 5.1. Пример Core Surface (MyExample.h)

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

@interface MainView : UIView
{
    CoreSurfaceBufferRef screenSurface;
    unsigned short *baseAddress;
    CALayer *screenLayer;
}

- (id)initWithFrame:(CGRect) frame;
- (void)drawRect:(CGRect) rect;
- (CoreSurfaceBufferRef) screenSurface;
- (void)dealloc;
@end

@interface MyApp : UIApplication
{
    UIWindow *window;
    MainView *mainView;
}

- (void)applicationDidFinishLaunching:(NSNotification *)aNotification;
@end
```

Листинг 5.2. Пример Core Surface (MyExample.m)

```
#import "MyExample.h"

int main(int argc, char **argv)
{
```

```
NSAutoreleasePool *autoreleasePool =
    [ [ NSAutoreleasePool alloc ] init ];
int returnCode = UIApplicationMain(argc, argv, @"MyApp", @"MyApp");
[ autoreleasePool release ];
return returnCode;
}

@implementation MyApp
- (void)applicationDidFinishLaunching:(NSNotification *)aNotification {
    window = [ [ UIWindow alloc ] initWithContentRect:
        [ UIHardware fullScreenApplicationContentRect ]
    ];

    CGRect rect = [ UIHardware fullScreenApplicationContentRect ];
    rect.origin.x = rect.origin.y = 0.0f;

    mainView = [ [ MainView alloc ] initWithFrame: rect ];

    [ window setContentView: mainView ];
    [ window orderFront: self ];
    [ window makeKey: self ];
    [ window _setHidden: NO ];

    NSTimer *timer = [ NSTimer scheduledTimerWithTimeInterval: 0.05
        target: self
        selector: @selector(handleTimer:)
        userInfo: nil
        repeats: YES ];
}

- (void) handleTimer: (NSTimer *) timer
{
    CoreSurfaceBufferRef screenSurface;
    unsigned short *baseAddress;
    int i;
```

```
screenSurface = [ mainView screenSurface ];
baseAddress = CoreSurfaceBufferGetBaseAddress(screenSurface);

for(i=0;i < 320 * 480;i++)
    baseAddress[i] = rand( ) % 0xFFFF;
[ mainView setNeedsDisplay ];
}
@end

@implementation MainView
- (id)initWithFrame:(CGRect)rect {

self = [ super initWithFrame: rect ];
if (nil != self) {
    CFMutableDictionaryRef dict;
    int x = 320, y = 480, pitch = x * 2, size = 2 * x * y, i;
    char *pixelFormat = "565L";

    /* Создаем поверхность экрана */
    dict = CFDictionaryCreateMutable(kCFAllocatorDefault, 0,
        &kCFTypedictionaryKeyCallbacks, &kCFTypedictionaryValueCallbacks);
    CFDictionarySetValue(dict, kCoreSurfaceBufferGlobal, kCFBooleanTrue);
    CFDictionarySetValue(dict, kCoreSurfaceBufferPitch,
        CFNumberCreate(kCFAllocatorDefault, kCFNumberSInt32Type, &pitch));
    CFDictionarySetValue(dict, kCoreSurfaceBufferWidth,
        CFNumberCreate(kCFAllocatorDefault, kCFNumberSInt32Type, &x));
    CFDictionarySetValue(dict, kCoreSurfaceBufferHeight,
        CFNumberCreate(kCFAllocatorDefault, kCFNumberSInt32Type, &y));
    CFDictionarySetValue(dict, kCoreSurfaceBufferPixelFormat,
        CFNumberCreate(kCFAllocatorDefault, kCFNumberSInt32Type,
            pixelFormat));
    CFDictionarySetValue(dict, kCoreSurfaceBufferAllocSize,
        CFNumberCreate(kCFAllocatorDefault, kCFNumberSInt32Type, &size));

    screenSurface = CoreSurfaceBufferCreate(dict);

```

```
screenLayer = [ [ CALayer layer ] retain ];
[ screenLayer setFrame: rect ];
[ screenLayer setContents: screenSurface ];
[ screenLayer setOpaque: YES ];

CoreSurfaceBufferLock(screenSurface, 3);
[ [ self _layer ] addSublayer: screenLayer ];

CoreSurfaceBufferUnlock(screenSurface);
}
return self;
}

- (void)drawRect:(CGRect)rect {
}
- (CoreSurfaceBufferRef)screenSurface {
return screenSurface;
}

- (void)dealloc
{
[ screenLayer release ];
[ self dealloc ];
[ super dealloc ];
}
@end
```

Как это работает

Пример с сумасшедшим снегом работает следующим образом:

1. При порождении приложения создается объект `MainView` и вызывается его метод `initWithFrame`.
2. `initWithFrame` создает 16-битную поверхность экрана размером 320×480 и привязывает ее к ее собственному объекту `CALayer`. Затем он добавляет этот уровень к основному виду.

3. Создается объект `NSTimer`, в результате чего каждые 0,05 секунд вызывается метод `handleTimer`. Эта процедура проходит по всей поверхности экрана и применяет случайное значение цвета к каждому пикселу. Кроме того, вызывается метод `setNeedsDisplay` данного объекта, приказывающий объекту перерисовать экран.

Анимация уровня

Хотя Quartz Core в предыдущем разделе и доказала свою полезность для стыковки буфера Core Surface с пользовательским интерфейсом, ее возможности гораздо шире, чем просто склеивающего уровня. Quartz Core может использоваться для трансформации объекта 2D в текстуру 3D, которая может быть использована для создания великолепных переходов между видами.

В главе 3 рассматривался класс `UITransitionView` как средство перехода между различными объектами `UIView`. Данный класс предлагал некоторые основные варианты анимации, но был двумерным. Платформа Quartz Core предоставляет более широкое множество инструментов для реализации анимации уровней для более эффектных переходов.

Создание перехода уровней

Переходы уровней усиливают существующий класс `UITransitionView`, предоставляя способ для переписывания его переходов новыми, использующими инструменты анимации Quartz Core. Это позволяет разработчику использовать преимущества расширенных возможностей 3D, предлагаемых Quartz Core, не внося при этом существенных изменений в свой код. Когда переход уровней, представляемый объектом `CATransition`, прикрепляется к `UITransitionView`, переход вызывает Quartz Core для порождения нового потока, который берет на себя всю графическую обработку перехода. Разработчику необходимо только добавить желаемый переход в свое приложение. В приведенном далее коде добавляется одна такая анимация:

```
CAAnimation *animation = [ CATransition animation ];
[ animation setType: @"pageCurl" ];
[ animation setSubtype: @"fromRight" ];
[ animation setTimingFunction:
    [ CAMediaTimingFunction functionWithName: @"easeInEaseOut" ] ];
```

```
[ animation setFillMode: @"extended" ];
[ animation setTransitionFlags: 3 ];
[ animation setSpeed: 0.25 ];
```

Доступные анимации

Предыдущий пример определяет анимацию `pageCurl` с помощью метода `setType`. В Quartz Core существуют следующие типы анимаций (табл. 5.1).

Таблица 5.1

Тип	Описание
<code>pageCurl</code>	Предыдущий вид закручивается, как листок вырывается из блокнота, открывая новый вид
<code>pageUnCurl</code>	Новый вид перелистывается поверх старого; обратный тип к <code>pageCurl</code>
<code>suckEffect</code>	Старый вид засасывается через нижний центр окна, открывая новый вид
<code>spewEffect</code>	Новый вид вырывается из нижнего центра окна; обратный тип к <code>suckEffect</code>
<code>genieEffect</code>	Старый вид засасывается через нижний левый или правый угол окна, открывая новый вид
<code>unGenieEffect</code>	Новый вид вырывается из нижнего левого или правого угла окна; обратный тип к <code>genieEffect</code>
<code>twist</code>	Вид вращается по горизонтали по типу спирального циклона
<code>tubey</code>	Вид вращается эластично по вертикали
<code>swirl</code>	Старый вид постепенно исчезает и заменяется новым видом, пока само окно поворачивается на 360°
<code>rippleEffect</code>	Новый вид волнами появляется в окне. Эта анимация не работает корректно при полноэкранных переходах
<code>cameraIris</code>	Шторка камеры закрывается на старом виде и открывается, представляя новый вид

Таблица 5.1 (окончание)

Тип	Описание
<code>cameraIrisHollow</code>	Аналогичен <code>cameraIris</code> , только старый вид исчезает до закрытия шторки
<code>cameraIrisHollowOpen</code>	Шторка камеры открывается только в новом виде; анимация начинается с закрытой шторкой
<code>cameraIrisHollowClose</code>	Шторка камеры закрывается на старом виде; анимация не открывает повторно шторку
<code>charminUltra</code>	Старый вид плавно и осторожно переходит в новый вид, не перегружая вас
<code>zoomyIn</code>	Новый вид "выезжает" на передний план сзади; старый вид уходит с переднего плана и исчезает
<code>zoomyOut</code>	Новый вид "выезжает" на передний план спереди; старый вид уходит назад
<code>oglApplicationSuspend</code>	Старый вид уходит назад; новый вид отображается незамедлительно. Похоже на нажатие домашней кнопки в приложении
<code>oglFlip</code>	Вид резко переворачивается по горизонтали, открывая новую страницу

Доступные подтипы

Чтобы задать направление, в котором должна происходить анимация, можно использовать приведенные в табл. 5.2 подтипы, которые задаются с помощью метода `setSubtype`.

Таблица 5.2

Тип	Описание
<code>fromLeft</code>	Анимация происходит слева направо
<code>fromRight</code>	Анимация происходит справа налево
<code>fromTop</code>	Анимация происходит сверху вниз
<code>fromBottom</code>	Анимация происходит снизу вверх

Скорость и хронометраж анимации

Скорость анимации имеет очень небольшой диапазон значений: от 0,0 (без движения) до примерно 5,0 (мгновенная). Самой большой приемлемой скоростью, скорее всего, будет 1,0.

Функция хронометража задает баланс между уходящей частью анимации и появляющейся частью анимации. Можно использовать различные временные промежутки, перечисленные в табл. 5.3.

Таблица 5.3

Тип	Описание
<code>easeInEaseOut</code>	На обе части анимации выделяется одинаковое время
<code>easeIn</code>	Вторая часть анимации выполняется быстрее
<code>easeOut</code>	Первая часть анимации выполняется быстрее

Флаги переходов

Флаги переходов определяют то, как сглаживаются границы. Значением является набор значений OR'd на основе макросов, перечисленных в табл. 5.4.

Таблица 5.4

Макрос	Значение	Положение бита
<code>kCALayerLeftEdge</code>	Сглаживание левых границ	$1 \ll 0$ (бит 0)
<code>kCALayerRightEdge</code>	Сглаживание правых границ	$1 \ll 1$
<code>kCALayerBottomEdge</code>	Сглаживание нижних границ	$1 \ll 2$
<code>kCALayerTopEdge</code>	Сглаживание верхних границ	$1 \ll 3$

Отображение перехода уровней

После того как анимация Quartz Core будет настроена, она выполняется посредством объекта `UITransitionView`. Прежде чем вызовется метод переход-

ного вида `transition`, анимация сначала должна привязаться к виду, который будет анимироваться. Чтобы анимировать все содержимое экрана, анимация может быть применена к основному виду. Это приведет к тому, что все объекты, принадлежащие виду, например, панели навигации и кнопки, будут включены в анимацию. Если вы работаете с небольшими объектами, например `UITextView` или `UIButton`, то примените анимацию к самому переходному виду. В результате этого будут задействованы только небольшие объекты:

```
[ [ transitionView _layer ] addAnimation: animation forKey: 0 ];
```

Эта строка добавляет анимацию к уровню так, что при его переходе анимация Quartz Core будет выполняться на месте перехода #0. Теперь осталось только одно — вызвать переход:

```
[ transitionView transition: 0 toView: newView ];
```

Пример: переворачивание страниц с применением стиля

В главе 3 для иллюстрации переходов вы создали программу переворачивания страниц. Мы воспользуемся тем же примером и здесь, только для переворачивания страниц добавим анимации Quartz Core.

Данный пример может быть скомпилирован с использованием пакета инструментов с помощью следующей командной строки:

```
$ arm-apple-darwin9-gcc -o MyExample MyExample.m -lobjc \
  -framework UIKit -framework CoreSurface \
  -framework CoreFoundation -framework QuartzCore \
  -framework Foundation
```

Листинги 5.3 и 5.4 содержат код для новой версии программы переворачивания страниц.

Листинг 5.3. Пример анимации Quartz Core (MyExample.h)

```
#import <CoreFoundation/CoreFoundation.h>
#import <UIKit/UIKit.h>
#import <UIKit/UINavigationController.h>
#import <UIKit/UINavigationControllerItem.h>
```

```
#import <UIKit/UITransitionView.h>
#import <UIKit/UITextView.h>
#import <QuartzCore/CATransition.h>
#import <QuartzCore/CAAnimation.h>
#define MAX_PAGES 10

@interface MainView : UIView
{
    UINavigationController *navBar;      /* Наша панель навигации */
    UINavigationControllerItem *navItem; /* Заголовок панели навигации */
    UITransitionView *transView;      /* Наш переход */
    int pageNum;                       /* Номер текущей страницы */

    /* Несколько страниц для прокрутки */
    UITextView *textPage[MAX_PAGES];
}

- (id)initWithFrame:(CGRect)frame;
- (void)dealloc;
- (UINavigationController *)createNavBar:(CGRect)rect;
- (void)setNavBar;
- (void)navigationBar:(UINavigationController *)navBar
    buttonClicked:(int)button;
- (void)flipTo:(int)page;

@end

@interface MyApp : UIApplication
{
    UIWindow *window;
    MainView *mainView;
}

- (void)applicationDidFinishLaunching:(NSNotification *)aNotification;
@end
```

Листинг 5.4. Пример анимации Quartz Core (MyExample.m)

```
#import "MyExample.h"

int main(int argc, char **argv)
{
    NSAutoreleasePool *autoreleasePool = [
        [NSAutoreleasePool alloc] init
    ];
    int returnCode = UIApplicationMain(argc, argv, @"MyApp", @"MyApp");
    [autoreleasePool release];
    return returnCode;
}

@implementation MyApp

- (void)applicationDidFinishLaunching:(NSNotification *)aNotification {
    UIWindow *window = [ [UIWindow alloc] initWithContentRect:
        [UIHardware fullScreenApplicationContentRect]
    ];

    CGRect rect = [UIHardware fullScreenApplicationContentRect];
    rect.origin.x = rect.origin.y = 0.0f;
    MainView *mainView = [ [MainView alloc] initWithFrame: rect ];

    [window setContentView: mainView];
    [window orderFront: self];
    [window makeKey: self];
    [window _setHidden: NO];
}

@end

@implementation MainView

- (id)initWithFrame:(CGRect)rect {
    self = [super initWithFrame: rect];
}
```

```
if (nil != self) {
    CGRect viewRect;
    int i;

    /* Создаем порт нового вида под панелью навигации */
    viewRect = CGRectMake(rect.origin.x, rect.origin.y + 48.0,
        rect.size.width, rect.size.height - 48.0);

    /* Задаем нашу начальную страницу */
    pageNum = MAX_PAGES / 2;

    /* Создаем десять объектов UITextView в качестве
        страниц в нашей книге */

    for(i=0;i<MAX_PAGES;i++) {
        textPage[i] = [ [ UITextView alloc ] initWithFrame: rect ];
        [ textPage[i] setText: [ [ NSString alloc ] initWithFormat:
            @"This is some text for page %d", i+1 ] ];
    }

    /* Создаем панель навигации с кнопками Prev и Next */
    navBar = [ self createNavBar: rect ];
    [ self setNavBar ];
    [ self addSubview: navBar ];

    /* Создаем наш переходной вид */
    transView = [ [ UITransitionView alloc ] initWithFrame: viewRect ];
    [ self addSubview: transView ];

    /* Переход к первой странице */
    [ self flipTo: pageNum ];
}

return self;
}
```

```
- (void)dealloc
{
    [ navBar release ];
    [ navItem release ];
    [ self dealloc ];
    [ super dealloc ];
}

- (UINavigationController *)createNavBar:(CGRect)rect {
    UINavigationController *newNav = [ [UINavigationController alloc] initWithFrame:
        CGRectMake(rect.origin.x, rect.origin.y, rect.size.width, 48.0f)
    ];

    [ newNav setDelegate: self ];
    [ newNav enableAnimation ];

    /* Добавляем наш заголовок */
    navItem = [ [UINavigationControllerItem alloc] initWithTitle:@"My Example" ];
    [ newNav pushViewController: navItem ];

    [ newNav showLeftButton:@"Prev" withStyle: 0
        rightButton:@"Next" withStyle: 0 ];

    return newNav;
}

- (void)setNavBar
{
    /* Делаем доступными или недоступными кнопки нашей страницы */
    if (pageNum == 1)
        [ navBar setButton: 1 enabled: NO ];
    else
        [ navBar setButton: 1 enabled: YES ];

    if (pageNum == MAX_PAGES)
        [ navBar setButton: 0 enabled: NO ];
}
```

```
else
    [ navbar setButton: 0 enabled: YES ];
}

- (void)navigationBar:(UINavigationController *)navbar buttonClicked:(int)button
{
    /* Следующая страница */
    if (button == 0)
    {
        [ self flipTo: pageNum+1 ];
    }

    /* Предыдущая страница */
    else {
        [ self flipTo: pageNum-1 ];
    }
}

- (void)flipTo:(int)page {

    CAAAnimation *animation = [ CATransition animation ];
    [ animation setType: @"ogFlip" ];
    [ animation setSubtype: @"fromLeft" ];
    [ animation setTimingFunction:
        [ CAMediaTimingFunction functionWithName: @"easeOut" ] ];
    [ animation setFillMode: @"extended" ];
    [ animation setTransitionFlags: 3 ];
    [ animation setSpeed: 0.50 ];
    [ [ self _layer ] addAnimation: animation forKey: 0 ];

    [ transView transition: 0 toView: textPage[page-1] ];
    pageNum = page;
    [ self setNavBar ];
}

@end
```

Как это работает

Данный пример работает совершенно аналогично примеру из листингов 3.9 и 3.10.

1. При инициализации приложения создается объект `MainView` и вызывается его метод `initWithFrame`. Тем самым создается десять объектов `UITextView`, которые станут примерами страниц нашей книги. Затем он создает панель навигации и один переходной вид. Наконец, вызывается метод `flipTo`, который отвечает за переворачивание страниц до указанного номера страницы.
2. Метод `flipTo` создает объект `CAAnimation` и добавляет его к уровню основного вида. Затем он вызывает переходной вид, чтобы проделать работу по запуску анимации, одновременно перемещаясь к требуемой странице. Анимация уровня основного вида используется автоматически, в результате чего осуществляется переход содержимого всего основного вида вместо простого переворачивания страницы.
3. Когда пользователь нажимает кнопку перехода `Prev` или кнопку перехода `Next`, то вызывается метод `buttonClicked` с указателем на панель навигации и номером нажатой кнопки. Здесь снова вызывается метод `flipTo` для перехода к новой странице и для того, чтобы сделать нажатую кнопку перехода недоступной, в случае, если достигнут конец книги.

Для дальнейшего изучения

Проверьте наличие прототипов `CALayer.h`, `CAAnimation.h` и `CATransition.h` в каталоге вашего пакета инструментов `include`. Вы найдете их в папке `/toolchain/sys/usr/include/QuartzCore`.

Преобразования уровней

Возможности Quartz Core по выводу на экран позволяют свободно манипулировать 2D-изображениями, как если бы они были 3D-изображениями. Изображение можно вращать на любой угол в осях x , y , z , масштабировать и наклонять. Объект `CATransform` — это магия под покровом технологии Cover Flow от Apple. Настольные системы Apple используют платформу Core Animation для обработки преобразований 3D. Платформа Quartz Core для iPhone использует многие подобные функции. iPhone поддерживает масштаб,

вращение, аффинные и параллельные преобразования. Более подробную информацию об этих различных преобразованиях можно найти в Core Animation Programming Guide от Apple, расположенном на Web-узле Apple Developer Connection.

Преобразование осуществляется для отдельных уровней. Платформа Quartz Core выполняет преобразования с помощью объекта CATransform. Этот объект применяется к уровню вида, чтобы повернуть или каким-либо другим способом трансформировать его уровень в требуемую конфигурацию 3D. Приложение же продолжает рассматривать данный объект как объект 2D, но при отображении пользователю этот объект соответствует тем преобразованиям, которые были применены к уровню. В приведенном далее примере создается преобразование для вращения уровня:

```
CATransform myTransform;
myTransform = CATransform3DMakeRotation(angle, x, y, z);
```

Метод CATransform3DMakeRotation создает преобразование, которое повернет уровень на количество радиан, определяемое `angle`, с использованием осей `x`, `y`, `z`. Значения `x`, `y`, `z` задают ось и величину каждого пространства (между -1 и $+1$). Присвоение значения какой-либо оси указывает преобразованию осуществить поворот с использованием именно этой оси. Например, если ось `x` установлена либо в -1 , либо в $+1$, то объект будет повернут по оси `x` в данном направлении, т. е. он будет повернут вертикально. Рассматривайте эти значения как вставку соломинок в изображение вдоль каждой оси. Если соломинка вставлена сквозь ось `x`, то изображение будет вращаться вокруг соломинки по вертикали. Более сложные повороты можно создавать с помощью значения углов осей. Однако в большинстве случаев значений -1 и $+1$ вполне достаточно.

Чтобы повернуть уровень на 45° по его горизонтальной оси (вертикальное вращение), можно использовать следующий код:

```
myTransform = CATransform3DMakeRotation(0.78, 1.0, 0.0, 0.0);
```

Чтобы повернуть уровень на 45° горизонтально, укажите соответствующее значение для оси `y`:

```
myTransform = CATransform3DMakeRotation(0.78, 0.0, 1.0, 0.0);
```

Использованное выше значение $0,78$ представляет радианное значение угла. Чтобы пересчитать градусы в радианы, воспользуйтесь простой формулой: $M\pi/180$. Например, $45\pi/180 = 45 \times 3,1415/180 = 0,7853$.

После создания преобразования оно применяется к соответствующему уровню. Чтобы получить доступ к этому уровню, воспользуйтесь методом `_layer` внутри любого объекта вида, чтобы получить его объект `CALayer`. Объект `CALayer` имеет метод `setTransform`, который вы используете для присоединения к нему преобразования. Тем самым указывается уровню выполнить заданное преобразование:

```
[ [ imageView _layer ] setTransform: myTransform ];
```

Пример: демонстрация вращения фонового рисунка

В данном примере используется преобразование `CATransform3DMakeRotate` из Quartz Core для вращения фонового рисунка рабочего стола несколькими различными способами по осям x , y и z . Фоновый рисунок загружается в класс `UIAutocorrectImageView`, который используется для уменьшения размера изображения в два раза (более детально это будет описано в главе 7). Затем используется таймер для осуществления вращения изображения каждые 0,01 секунд и изменением оси в конце каждого поворота на 360° .

Чтобы скомпилировать данный пример, воспользуйтесь пакетом инструментов в командной строке:

```
$ arm-apple-darwin9-gcc -o MyExample MyExample.m -lobjc \
-framework Foundation -framework CoreFoundation -framework UIKit \
-framework QuartzCore
```

В листингах 5.5 и 5.6 приведен соответствующий код.

Листинг 5.5. Пример преобразования уровня (MyExample.h)

```
#import <CoreFoundation/CoreFoundation.h>
#import <UIKit/UIKit.h>
#import <QuartzCore/QuartzCore.h>
#import <QuartzCore/CATransform.h>
```

```
@interface MainView : UIView
{
    UIAutocorrectImageView *imageView;
}
```

```
    CATransform transform;
    NSTimer *timer;

    float angle, x, y, z;
}

- (id)initWithFrame:(CGRect) frame;
- (void) handleTimer: (NSTimer *) timer;
- (void) dealloc;

@end

@interface MyApp : UIApplication
{
    UIWindow *window;
    MainView *mainView;
}

- (void) applicationDidFinishLaunching: (NSNotification *) aNotification;
@end
```

Листинг 5.6. Преобразование уровня (MyExample.m)

```
#import <UIKit/UIKit.h>
#import <UIKit/UIAutocorrectImageView.h>
#import "MyExample.h"

int main(int argc, char **argv)
{
    NSAutoreleasePool *autoreleasePool = [
        [ NSAutoreleasePool alloc ] init
    ];
    int returnCode = UIApplicationMain(argc, argv, @"MyApp", @"MyApp");
    [ autoreleasePool release ];
    return returnCode;
}
```

```
@implementation MyApp

- (void)applicationDidFinishLaunching:(NSNotification *)aNotification {
    window = [ [ UIWindow alloc ] initWithContentRect:
        [ UIHardware fullScreenApplicationContentRect ]
    ];

    CGRect rect = [ UIHardware fullScreenApplicationContentRect ];
    rect.origin.x = rect.origin.y = 0.0f;

    mainView = [ [ MainView alloc ] initWithFrame: rect ];

    [ window setContentView: mainView ];
    [ window orderFront: self ];
    [ window makeKey: self ];
    [ window _setHidden: NO ];
}

@end

@implementation MainView

- (id)initWithFrame:(CGRect)rect {

    self = [ super initWithFrame: rect ];
    if (nil != self) {
        UIImage *tempImage;

        angle = y = z = 0;
        x = 1;

        tempImage = [ UIImage defaultDesktopImage ];
        imageView = [ [ objc_getClass("UIAutocorrectImageView") alloc ]
            initWithFrame: CGRectMake(80, 120, 160, 240)
            image: tempImage ];
        [ [ self _layer ] addSublayer: [ imageView _layer ] ];
    }
}
```

```
transform = CATransform3DMakeRotation(angle, x, y, z);
[ imageView_layer ].transform = transform;

[ self setNeedsDisplay ];

timer = [ NSTimer scheduledTimerWithTimeInterval: 0.01
        target: self
        selector: @selector(handleTimer:)
        userInfo: nil
        repeats: YES ];
}
return self;
}

- (void) handleTimer: (NSTimer *) timer
{
    angle += 0.01;
    if (angle > 6.283) {
        angle = 0;
        if (z == 1) {
            x = 0;
        }
        else {
            if (y == 1) {
                z = 1;
            }
            y = 1;
        }
    }

    [ [ imageView_layer ] setTransform:
      CATransform3DRotate(transform, angle, x, y, z)
    ];
}
```

```
- (void)dealloc
{
    [ imageView release ];
    [ self dealloc ];
    [ super dealloc ];
}
@end
```

Как это работает

Демонстрация преобразования работает следующим образом:

1. При порождении приложения создается объект `MainView` и вызывается его метод `initWithFrame`.
2. В результате инициализируется угол осей x , y , z , и в `UIAutocorrectImageView` загружается фоновый рисунок рабочего стола, где его размер уменьшается в половину. Данное изображение отображается в середине экрана.
3. Запускается таймер, чтобы каждые 0,01 секунд вызывать метод `handleTimer`. С каждым тиком таймера угол увеличивается и после достижения 360° (6,283 радиан) переворачивается.
4. С помощью метода `CATransform3DRotate` преобразование модифицируется и применяется к уровню. Это приводит к ощущению вращения изображения в центре экрана.
5. При каждом перевороте угла устанавливаются различные оси, меняя тем самым направление, в котором вращается объект.

Для дальнейшего изучения

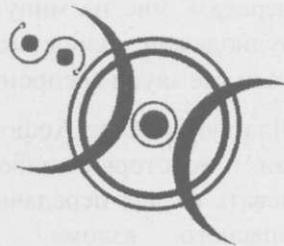
Чуть глубже изучите преобразования и попробуйте выполнить следующие упражнения.

- Проверьте наличие прототипов `CATransform.h` и `CALayer.h` в каталоге вашего пакета инструментов `include`. Вы найдете их в папке `/toolchain/sys/usr/include/QuartzCore`.

- Посетите http://en.wikipedia.org/wiki/Axis_angle, где объясняется, как работают осевые углы. Это может пригодиться при разработке собственных преобразований вращения.
- Изучите Apple's Core Animation guide на Web-узле Apple Developer Connection. Функции Core Animation отображают функции Quartz Core в iPhone. Это руководство можно найти по адресу:

http://developer.apple.com/documentation/Cocoa/Conceptual/CoreAnimation_guide/.

ГЛАВА 6



Звук

Если и существовали какие-либо сомнения, является ли iPhone устройством для воспроизведения музыки, то они полностью развеяны наличием на нем трех разных платформ исключительно для звуковых эффектов. Платформы Core Audio, Celestial и Audio Toolbox предоставляют различные уровни функциональности для работы со звуком. Кроме этого, на iPhone выполняется аудиодемон (audio daemon) mediaserverd, который объединяет звуковой вывод всех приложений и управляет такими событиями, как изменение уровня громкости и состояний переключателя звонка. Платформа iPhone для работы со звуком имеет множество нюансов, но Apple предоставила великолепные интерфейсы, существенно облегчающие работу со звуком.

Core Audio: великолепна, но вы не можете ее использовать

Среди всех трех имеющихся платформ Core Audio является наиболее низкоуровневой и наименее доступной. Core Audio предоставляет прямой интерфейс к звуковому устройству iPhone. Поскольку существует только одно звуковое устройство, то в любой момент времени общаться с ним может только один процесс. В отличие от настольной системы Mac OS X, которая позволяет разработчикам разделять ресурсы Core Audio, в iPhone имеется аудиодемон, который привязывается к устройству, как только iPhone загрузится, устанавливая то, что во многих версиях фирменного программного обеспечения называется *hog mode*. *Hog mode* — это флаг, жестко запрограммированный в платформу Core Audio, лишаящий любое другое приложение возможности сказать: "Эй, я бы хотел издать некоторые звуки, не мог бы ты

передать мне на минутку управление звуковой картой?" Другими словами, аудиодемон полностью захватывает звуковую карту себе, заставляя тем самым все звуки воспроизводиться посредством самого демона, а не напрямую.

Платформа Core Audio была первой платформой, разгаданной разработчиками, и некоторые наиболее ранние приложения для iPhone пытались использовать ее для передачи потока цифрового звука. Это требовало ужасного и опасного взлома, принуждавшего пользователя убивать процесс `mediaserverd`. Сама программа могла потом быть услышанной, но подобный взлом убивал весь остальной звук на iPhone. Многие пользователи настолько страстно желали играть на iPhone в видеоигры, что прибегали к подобному взлому. Но теперь, когда обнаружены платформы Celestial и Audio Toolbox, больше нет необходимости использовать Core Audio, и обновлены все приложения, которые раньше ее использовали.

Если вы все же хотите изучить платформу Core Audio, то для вас есть хорошая новость: она полностью идентична настольной версии. Множество ресурсов, относящихся к платформе Core Audio, имеется на Web-узле Apple Developer Connection, расположенном по адресу: <http://developer.apple.com/audio/>.

Celestial

Celestial — это платформа iPhone, предпочтительная для воспроизведения звуковых и музыкальных файлов, а также для записи звука со встроенного микрофона. Для воспроизведения звуковых дорожек, представленных в виде объектов `AVItem`, Celestial использует класс `AVController`. Данная платформа также поддерживает необязательный класс `AVQueue` для упорядочивания воспроизведения различных звуковых дорожек.

То, для чего Celestial совершенно бесполезна, так это воспроизведение цифрового потока, т. е. необработанного выходного канала звука. Это будет описано в следующем разделе об Audio Toolbox. Celestial работает исключительно с аудиофайлами.

Чтобы приступить к работе с Celestial, ваше приложение должно подключиться к платформе Celestial. С помощью пакета инструментов подключите Celestial к вашему приложению, добавив следующие аргументы к аргументам компилятора, описанным в главе 2:

```
$ arm-apple-darwin9-gcc -o MyApp MyApp.m -lobjc \
    -framework CoreFoundation
```

```
-framework Foundation  
-framework Celestial
```

Чтобы добавить это в пример make-файла из главы 2, добавьте платформу Celestial в раздел флагов компоновщика, подключив тем самым данную библиотеку:

```
LDFLAGS = -lobjc  
-framework CoreFoundation  
-framework Foundation  
-framework Celestial
```

Метод *ringerState*

Воспроизведение звуков может быть очень полезным для приложений различных типов, но хорошим тоном считается, если разработчик убедится в том, не пожелал ли пользователь отключить звук своего телефона. Прежде чем звук будет воспроизведен, вы должны проверить состояние звонка — переключатель подавления звука на телефоне. Для этого UIKit предоставляет в рамках класса `UIHardware` метод `ringerState`:

```
int ringerState = [ UIHardware ringerState ];
```

Если переключатель подавления звука установлен в положение "звучать", то метод вернет значение 1. Если же переключатель подавления звука установлен в положение "подавлять", то метод вернет значение 0. Если звонок отключен, то вместо звуковых уведомлений принято использовать встроенный в телефон вибратор. В приложении можно найти пример такого использования. Подавлять звук необязательно, т. е. операционная система не принуждает этого делать, поскольку некоторые приложения предназначались для генерирования звука независимо от состояния переключателя звонка. Например, приложение iPod воспроизводит музыку, даже если звонок отключен, позволяя тем самым пользователю слушать песню, не отвлекаясь на телефонные звонки (например, если он или она занимается пробежкой).

Аудиоконтроллер

Класс `AVController` устанавливает соединение с аудиодемоном iPhone. Кроме того, он отвечает за управление всем воспроизводимым через него звуком.

AVController можно рассматривать как фильтр, расположенный внизу вашего приложения iPod (панель, содержащая элементы управления воспроизведением, приостановкой, перемещением и уровнем громкости). Данный класс управляет всеми этими функциями, а также настройками эквалайзера, частотой звука и даже режимом повторения:

```
AVController * av = [ [ AVController alloc ] init ];
```

Уровень громкости

Уровень громкости (volume) может быть настроен как для отдельной звуковой дорожки, так и для всего контроллера. Чтобы настроить его в контексте контроллера, вызовите метод AVController класса `setVolume:`:

```
[ av setVolume: 1.0 ]
```

Уровень громкости является числом с плавающей запятой от 0,0 до 1,0. Для настройки громкости на уровне контроллера пользователю будет отображено диалоговое окно. Это очень удобно для управления запросами на подавление звука и т. п.

Настройка уровня громкости для отдельной звуковой дорожки будет показана в разд. "Аудиодорожки" далее в этой главе.

Режим повторения

Чтобы указать контроллеру повторять звук после завершения воспроизведения, вызовите `setRepeatMode`, передав в качестве аргумента целое значение:

```
[ av setRepeatMode: 1 ];
```

Режим повторения (repeat mode) должен настраиваться после того, как начато воспроизведение звуковой дорожки. Существуют следующие режимы повторения:

- 0 — повторение отключено;
- 1 — повторение включено.

Несмотря на то, что режим повторения настраивается на уровне контроллера, повторяться будет только текущая звуковая дорожка, даже если в очереди на воспроизведение находятся еще и другие дорожки.

Частота звуковой дорожки

Частота звуковой дорожки (sample rate) — это частота воспроизведения, на которой должны воспроизводиться звуки. Она должна соответствовать той, на которой были первоначально записаны звуковые дорожки, и может быть установлена как вручную с помощью показанного здесь метода `setRate`, так и автоматически, не затрагивая этот параметр:

```
NSError *err;
[ av setRate: 44100.0 error: &err ];
if (err != nil) {
    NSLog(@"The following error has occurred: %@", err);
}
```

Многие методы `Celestial` должны уметь сообщать об ошибке. Для этого применяется класс `NSError`. Он похож на объект `NSString`, являющийся стандартным для обеих операционных систем: iPhone и Mac OS X. Этот класс построен на базовом классе `NSString` и инкапсулирует дополнительную информацию, имеющую отношение к кодам ошибок. Более подробную информацию о `NSError` и `NSString` можно найти в справочнике, расположенном на Web-узле `Apple Developer Connection`.

Заготовки эквалайзера

Эквалайзер настраивает относительный уровень громкости различных частот для воспроизведения более чистого звука. Различные эквалайзеры улучшают качество воспроизводимых пользователем различных типов записей и музыки, и Apple использует заготовки эквалайзера (equalizer presets), широко применяемые в iTunes и последующих продуктах. iPhone поддерживает 22 различных заготовки эквалайзера, которые могут быть выбраны с помощью метода `setEQPreset`:

```
[ av setEQPreset: 0 ];
```

Настройка по умолчанию задает использование эквалайзера, и если вы воспроизводите только звуки событий, то нет особого смысла это менять. Однако если же вы прослушиваете радиопередачи или записываете звуки с помощью проигрывателя сторонних фирм, то для выбора желаемой заготовки эквалайзера можно воспользоваться табл. 6.1.

Таблица 6.1

Заготовка эквалайзера	Описание
0	Отключено
1	Акустика (Acoustic)
2	Усилитель низких частот (Bass Booster)
3	Понижатель низких частот (Bass Reducer)
4	Классика (Classical)
5	Танцевальная (Dance)
6	Низкая (Deep)
7	Электронная (Electronic)
8	Бемоль (Flat)
9	Хип-хоп (Hip Hop)
10	Джаз (Jazz)
11	Латина (Latin)
12	Громкость (Loudness)
13	Неторопливая (Lounge)
14	Фортепьяно (Piano)
15	Поп (Pop)
16	R&B
17	Рок (Rock)
18	Небольшие динамики (Small Speakers)
19	Произносимое слово (Spoken Word)
20	Усилитель верхних частот (Treble Booster)
21	Понижатель верхних частот (Treble Reducer)
22	Усилитель вокала (Vocal Booster)

Подавление звука

Подавить аудиоканал так же просто, как отправить сообщение методу `setMuted`. Данный метод применяется не только к текущему воспроизводимому звуку, но и к каждому звуку, воспроизводимому через объект контроллера после него:

```
[ av setMuted: NO ];
```

Аудиодорожки

Итак, вы породили объект `AVController`, который создает аудиоканал, но до сих пор еще ничего не было сделано для воспроизведения звуковых дорожек. Создайте каждую воспроизводимую дорожку как объект `AVItem`. Приведенный далее фрагмент кода создает `AVItem` для ссылки на существующий в iPhone звуковой файл. Для отлавливания ошибок здесь также используется класс `NSError`, с которым вы познакомились в последнем разделе:

```
NSError *err;
AVItem *item = [ [ AVItem alloc ]
    initWithPath: @"~/Library/Ringtones/Pinball.m4r" error: &err
];
if (err != nil) {
    NSLog(@"The following error has occurred: %@", err);
}
```

Воспроизведение URL

С помощью того же класса `AVItem` можно воспроизводить URL, указывая URL вместо пути файла:

```
AVItem *item = [ [ AVItem alloc ]
    initWithPath: @"http://path-to-sound-file" error: &err
];
```

Это может пригодиться, но имейте в виду, что для воспроизведения этих звуков вашему приложению потребуется сетевое подключение. Медленное подключение или его полное отсутствие может привести к замедлению работы приложения.

Уровень громкости звуковой дорожки

Чтобы установить уровень громкости для какой-либо отдельной звуковой дорожки, можно вызвать метод `setVolume` класса `AVItem`. В отличие от метода контроллера использование здесь `setVolume` не повлияет на остальные аудиоканалы, а также пользователю не будет отображаться диалоговое окно изменения уровня громкости:

```
[ item setVolume: 0.5 ];
```

Заготовки эквалайзера

Отдельно взятая звуковая дорожка также обладает собственным свойством EQ, которое может быть настроено с помощью табл. 6.1. Эта настройка EQ распространяется только на данную конкретную дорожку и остается действительной до тех пор, пока существует данный объект:

```
[ item setEQPreset: 0 ];
```

Длительность

После того как объект будет создан, вы можете определить длительность звуковой дорожки в секундах:

```
float duration = [ item duration ];
```

Аудиоочереди

Чтобы воспроизводить звук через аудиоконтроллер, вам необходим способ помещать звуковые дорожки в очередь. Воспроизведение двух звуков подряд без очереди невозможно, поскольку тогда непонятно, как сообщить объекту контроллера сделать это, не убивая при этом воспроизводимый в данный момент звук. Аудиоочереди предоставляют решение этой проблемы путем создания структуры наподобие массива, в которую можно помещать звуковые дорожки для упорядоченного воспроизведения.

В данном разделе рассматриваются аудиоочереди, имеющиеся в классе контроллера `AVController`, который воспроизводит предварительно записанные аудиовыборки из файлов. Если какое-либо приложение генерирует собственные звуки, то оно должно использовать аудиоочередь `Audio Toolbox`, описываемую далее в этой главе.

Очередь, предоставляемая классом аудиоконтроллера, — это класс `AVQueue`:

```
AVQueue *avq = [ [ AVQueue alloc ] init ];
```

Аудиоочередь прикрепляется к аудиоконтроллеру, а контроллер признает ее в качестве источника всех будущих воспроизведений:

```
[ av setQueue: avq ];
```

Звуковые дорожки, порожденные как объекты `AVItem`, затем могут быть добавлены, удалены и переупорядочены в очереди с помощью целого ряда существующих методов:

- добавить дорожку в конец очереди

```
[ avq appendItem: item ];
```

- вставить дорожку после другой дорожки, определяемой объектом `AVItem` другой дорожки

```
[ avq insertItem: item afterItem: other_item error: &err ];
```

- вставить дорожку в определенное место в очереди

```
[ avq insertItem: item atIndex: 4 error: &err ];
```

- удалить дорожку, определяемую объектом `AVItem` дорожки

```
[ avq removeItem: item ];
```

- удалить дорожку, определяемую местом в очереди

```
[ avq removeItemAtIndex: 3 ];
```

- удалить все элементы в пределах заданного диапазона мест в очереди

```
[ avq removeItemsInRange: NSRange(3, 4) ];
```

Функция `NSMakeRange` принимает два аргумента: начальное местоположение и длину диапазона. Приведенный пример удаляет дорожки 3—7;

- удалить все аудиодорожки из очереди

```
[ avq removeAllItems ];
```

- когда контроллеру будет отдано указание начать воспроизведение, он будет пошагово перемещаться в очереди от элемента к элементу:

```
[ av play: nil ];
```

Метод воспроизведения вызывается таким же образом, как и в случае отсутствия очереди, но поскольку очередь прикреплена к контроллеру, то он будет использовать ее в качестве источника дорожек без каких-либо особых указаний. Во время воспроизведения элементы в очереди можно добавлять, уда-

лять или переупорядочивать до тех пор, пока очередь не дойдет до задействованных в перемещениях элементов.

Пример: переменные мелодии звонка

В данном примере будут воспроизводиться одна за другой две отдельные мелодии звонка посредством аудиоочереди.

Скомпилируйте этот пример с помощью следующей командной строки:

```
$ arm-apple-darwin9-gcc -o MyExample MyExample.m -lobjc \
    -framework Foundation \
    -framework CoreFoundation \
    -framework UIKit \
    -framework Celestial
```

В листингах 6.1 и 6.2 приведен соответствующий код.

Листинг 6.1. Пример аудиоочереди (MyExample.h)

```
#import <CoreFoundation/CoreFoundation.h>
#import <Foundation/Foundation.h>
#import <UIKit/UIKit.h>
#import <UIKit/UITextView.h>
#import <Celestial/AVController.h>
#import <Celestial/AVItem.h>
#import <Celestial/AVQueue.h>

@interface MainView : UIView
{
    UITextView *textView;
    AVController *av;
    AVItem *item1, *item2;
    AVQueue *avq;
}

- (id)initWithFrame:(CGRect) frame;
- (void)dealloc;
@end
```

```
@interface MyApp : UIApplication
{
    UIWindow *window;
    MainView *mainView;
}

- (void)applicationDidFinishLaunching:(NSNotification *)aNotification;
@end
```

Листинг 6.2. Пример аудиоочереди (MyExample.m)

```
#import "MyExample.h"

int main(int argc, char **argv)
{
    NSAutoreleasePool *autoreleasePool = [
        [ NSAutoreleasePool alloc ] init
    ];
    int returnCode = UIApplicationMain(argc, argv, @"MyApp", @"MyApp");
    [ autoreleasePool release ];
    return returnCode;
}

@implementation MyApp

- (void)applicationDidFinishLaunching:(NSNotification *)aNotification {
    window = [ [ UIWindow alloc ] initWithContentRect:
        [ UIHardware fullScreenApplicationContentRect ]
    ];

    CGRect rect = [ UIHardware fullScreenApplicationContentRect ];
    rect.origin.x = rect.origin.y = 0.0f;

    mainView = [ [ MainView alloc ] initWithFrame: rect ];
}
```

```
[ window setContentView: mainView ];
[ window orderFront: self ];
[ window makeKey: self ];
[ window _setHidden: NO ];
}

@end

@implementation MainView
- (id)initWithFrame:(CGRect)rect {

self = [ super initWithFrame: rect ];
if (nil != self) {
    NSError *err;
    textView = [ [ UITextView alloc ] initWithFrame: rect ];
    [ textView setText: @"Hello, Sound!" ];
    [ self addSubview: textView ];

    av = [ [ AVController alloc ] init ];
    avq = [ [ AVQueue alloc ] init ];

    item1 = [ [ AVItem alloc ]
        initWithPath:@"~/Library/Ringtones/Pinball.m4r" error:&err
    ];
    if (err != nil)
        [ textView setText: err ];

    item2 = [ [ AVItem alloc ]
        initWithPath:@"~/Library/Ringtones/Blues.m4r" error: &err
    ];
    if (err != nil)
        [ textView setText: err ];

    [ avq appendItem: item1 error: &err ];
    [ avq appendItem: item2 error: &err ];

```

```
[ av setQueue: avq ];  
[ av play:nil ];  
}  
return self;  
}  
- (void)dealloc  
{  
[ self dealloc ];  
[ super dealloc ];  
}  
@end
```

Как это работает

Пример работает следующим образом:

1. Порождается приложение и отображается текстовое поле с заданным по умолчанию текстом: "Hello, Sound!"
2. Порождаются объекты AVController и AVQueue.
3. Вместо создания единичного объекта AVItem создаются два: один — для мелодии звонка Pinball, а другой — для мелодии звонка Blues, оба расположенные в папке /Library/Ringtones.
4. Если любой из объектов AVItem вызовет ошибку, то текст в текстовом виде заменяется сообщением об ошибке.
5. Обе дорожки добавляются в объект аудиоочереди.
6. Объект аудиоочереди прикрепляется к контроллеру, а контроллеру дается указание начать воспроизведение.
7. Обе аудиодорожки воспроизводятся посредством очереди в том порядке, в котором они были в нее добавлены.

Для дальнейшего изучения

Чтобы быть более спокойными относительно Celestial, проверьте наличие AVController.h, AVItem.h и AVQueue.h в вашем пакете инструментов. Вы обнаружите их в папке /toolchain/sys/usr/include/Celestial.

Audio Toolbox

Платформа Audio Toolbox является новой для Leopard и доступна для настольных систем и iPhone. Audio Toolbox как расширение Core Audio предоставляет множество низкоуровневых функций для обработки, воспроизведения и записи звука на уровне потока битов. Платформа включает множество API, обеспечивающих доступ к необработанным данным аудиофайлов и множество инструментов конвертации.

В отличие от многих уже описанных в этой книге платформ, платформа Audio Toolbox является более C-ориентированной. На Web-узле Apple Developer Connection имеется целый ряд справочных материалов, посвященных Audio Toolbox. Например:

- ❑ Core Audio Overview: Audio Toolbox framework
http://developer.apple.com/documentation/MusicAudio/Conceptual/CoreAudioOverview/Introduction/chapter_1_section_1.html
- ❑ Audio Toolbox Framework Reference
<http://developer.apple.com/DOCUMENTATION/MusicAudio/Reference/CAAudioTooboxRef/index.html>
- ❑ Audio File Services Reference
<http://developer.apple.com/documentation/MusicAudio/Reference/AudioFileConvertRef/Reference/reference.html>
- ❑ Audio File Stream Services Reference
<http://developer.apple.com/documentation/MusicAudio/Reference/AudioStreamReference/AudioStreamReference.pdf>
- ❑ Audio Queue Services Reference
<http://developer.apple.com/documentation/MusicAudio/Reference/AudioQueueReference/AudioQueueReference.pdf>

Поскольку платформа Audio Toolbox существует и для настольных систем, то она достаточно неплохо документирована. Здесь мы не собираемся описывать ее целиком, а уделим внимание только тем ее частям, которые относятся к iPhone. Многие составляющие данной платформы, например, MIDI-контроллеры и Music Player API, несущественны или же вовсе недоступны для iPhone. Чтобы начать использование Audio Toolbox, вам потребуется

скопировать заголовки Audio Toolbox с вашего рабочего стола в пакет инструментов:

```
$ sudo cp -pr /System/Library/Frameworks/AudioToolbox.framework/Headers  
/toolchain/sys/usr/include/AudioToolbox
```

Также в вашем make-файл вам нужно будет добавить следующие флаги компиляции:

```
-DMAC_OS_X_VERSION_MAX_ALLOWED=MAC_OS_X_VERSION_10_5  
-DMAC_OS_X_VERSION_MIN_REQUIRED=MAC_OS_X_VERSION_10_5
```

"Другая" аудиоочередь: для звука, генерируемого приложением

Класс `AVQueue` из `Celestial`, описанный в последнем разделе, подходит для постановки в очередь самосодержащих, заранее записанных аудиодорожек. Но сообщество разработчиков iPhone не раз озвучивало настоятельные требования добавления инструментов, которые могли бы воспроизводить аудиопотоки, генерируемые приложениями в процессе своей работы, например, играми.

Такие приложения могут использовать Audio Toolbox, которая обладает собственной реализацией аудиоочереди, созданной для необработанных звуковых данных. Это весьма полезно для приложений, которые порождают свой непрерывный поток цифрового звука. Очередь Audio Toolbox является совершенно независимой от окружения контроллера Celestial и работает с потоками необработанных аудиоданных, а не с законченными файлами.

Аудиоочередь можно рассматривать как ленточный конвейер, заполненный ящиками. На одном конце конвейера ящики под завязку заполнены звуком, а на другом конце они загружаются в динамики iPhone. Эти ящики символизируют звуковые буферы, переносящие биты, а ленточный конвейер символизирует аудиоочередь. Конвейер выгружает ваш звук в динамики, а затем возвращается по кругу, чтобы снова заполнить ящики. Ваша работа, как программиста, состоит в том, чтобы определить размер, тип и количество ящиков, и написать программное обеспечение для заполнения ящиков необходимым вам звуком.

В отличие от очереди Celestial, очередь Audio Toolbox строго придерживается принципа "первым прибыл — первым обслужен" (first in, first out). В то время как очередь Celestial позволяет вам переупорядочивать аудиодорожки

в очереди, конвейер Audio Toolbox воспроизводит дорожки строго в том порядке, в котором они были добавлены.

Аудиоочередь Audio Toolbox работает следующим образом:

1. Создается аудиоочередь, и задаются свойства, определяющие тип звука, который будет воспроизводиться (формат, частота дорожки и т. д.)
2. К очереди прикрепляются звуковые буферы, которые будут содержать сами звуковые фреймы для воспроизводства. Звуковой фрейм можно рассматривать как отдельный ящик, заполненный звуком, а дорожку — как отдельную часть цифрового звука внутри фрейма.
3. Разработчик предоставляет функцию обратного вызова, которая вызывает аудиоочередь при каждом опустошении звукового буфера. Тем самым происходит повторное заполнение буфера самыми последними звуковыми фреймами из вашего приложения.

Структура аудиоочереди

Поскольку платформа Audio Toolbox использует низкоуровневые интерфейсы C, то в ней нет понятия класса. Существует множество движущихся частей, вовлеченных в подготовку аудиоочереди, и чтобы сделать наши примеры более понятными, все эти используемые различные переменные будут инкапсулированы в единую, задаваемую пользователем структуру, которую мы назовем `AQCallbackStruct`:

```
typedef struct AQCallbackStruct {  
    AudioQueueRef queue;  
    UInt32 frameCount;  
    AudioQueueBufferRef mBuffers[AUDIO_BUFFERS];  
    AudioStreamBasicDescription mDataFormat;  
} AQCallbackStruct;
```

Следующие компоненты сгруппированы в данную структуру для обслуживания аудиоплатформы:

- `AudioQueueRef queue` — указатель на объект аудиоочереди, который создаст ваша программа;
- `UInt32 frameCount` — общее число дорожек, копируемых за одну аудиосинхронизацию. Это полностью отдается на откуп реализующему;

- `AudioQueueBufferRef mBuffers` — массив, содержащий общее число звуковых буферов, которые будут использоваться. Точное количество элементов будет обсуждаться в разд. "Звуковые буферы" далее в этой главе;
- `AudioStreamBasicDescription mDataFormat` — информация о формате аудио, которое будет воспроизводиться.

Прежде чем аудиоочередь может быть создана, вы обязаны инициализировать экземпляры следующих переменных:

```
AQCallbackStruct aqc;
aqc.mDataFormat.mSampleRate = 44100.0;
aqc.mDataFormat.mFormatID = kAudioFormatLinearPCM;
aqc.mDataFormat.mFormatFlags = kLinearPCMFormatFlagIsSignedInteger
    | kAudioFormatFlagIsPacked;
aqc.mDataFormat.mBytesPerPacket = 4;
aqc.mDataFormat.mFramesPerPacket = 1;
aqc.mDataFormat.mBytesPerFrame = 4;
aqc.mDataFormat.mChannelsPerFrame = 2;
aqc.mDataFormat.mBitsPerChannel = 16;
aqc.frameCount = 735;
```

В данном примере мы подготавливаем структуру для 16-битного (два байта на дорожку) стереозвуча (два канала) с частотой дорожки 44 кГц (44 100). Наша выходная дорожка будет представлена в форме двух двухбайтных коротких целых чисел, а, следовательно, четыре полных байта на фрейм (по два байта для левого и правого каналов).

Частота дорожки и размер фрейма определяют то, как часто iPhone будет запрашивать звук. С частотой в 44 100 дорожки в секунду мы можем заставить наше приложение синхронизировать звук каждую 1/60-ую часть секунды, определив размер фрейма в 735 дорожек ($44\ 100/60 = 735$).

Формат, используемый в этом примере, — PCM (необработанные данные), но iPhone поддерживает и другие форматы:

- `kAudioFormatLinearPCM`;
- `kAudioFormatAppleIMA4`;
- `kAudioFormatMPEG4AAC`;
- `kAudioFormatULaw`;
- `kAudioFormatALaw`;

- `kAudioFormatMPEGLayer3`;
- `kAudioFormatAppleLossless`;
- `kAudioFormatAMR`.

Подготовка аудиовывода

После того как свойства аудиоочереди будут определены, можно подготовить объект новой аудиоочереди. За подготовку выходного канала и прикрепление его к очереди отвечает функция `AudioQueueNewOutput`. Прототип функции выглядит следующим образом:

```
AudioQueueNewOutput (
    const AudioStreamBasicDescription *inFormat,
    AudioQueueOutputCallback inCallbackProc,
    void * inUserData,
    CFRunLoopRef inCallbackRunLoop,
    CFStringRef inCallbackRunLoopMode,
    UInt32 inFlags,
    AudioQueueRef * outAQ);
```

Здесь:

- `inFormat` — указатель на структуру, описывающую аудиоформат, который будет воспроизводиться. Мы определили эту структуру ранее как члена типа данных `AudioStreamBasicDescription` в рамках нашей структуры `AQCallbackStruct`;
- `inCallbackProc` — имя функции обратного вызова, которая будет вызываться, когда буфер аудиоочереди окажется пустым и потребует данных;
- `inUserData` — указатель на данные, которые разработчик при желании может передавать в функцию обратного вызова. Он будет содержать указатель на экземпляр определяемой пользователем структуры `AQCallbackStruct`, которая должна содержать информацию как об аудиоочереди, так и любую другую относящуюся к приложению информацию о воспроизводимых дорожках;
- `inCallbackRunLoopMode` — сообщает аудиоочереди о том, какое должно быть зацикливание аудио. Если указан `NULL`, то функция обратного вызова вызывается каждый раз, когда опустошается звуковой буфер. Для запуска обратного вызова при других условиях существуют дополнительные модели;

- `inFlags` — не используется, зарезервировано;
- `outAO` — когда функция `AudioQueueNewOutput` возвращает результат, данный указатель устанавливается на только что созданную аудиоочередь. Присутствие данного аргумента позволяет коду ошибки использоваться в качестве возвращаемого значения данной функции.

Действительный вызов данной функции с использованием созданной ранее структуры аудиоочереди выглядит следующим образом:

```
AudioQueueNewOutput(&aqc.mDataFormat, AQueueCallback, &aqc,  
                  NULL, kCFRunLoopCommonModes, 0, &aqc.queue);
```

В данном примере имя функции обратного вызова определено как `AQueueCallback`. Эта функция будет создана в нескольких следующих разделах. Это функция, которая будет отвечать за прием звукового вывода из вашего приложения и копирование его в звуковой буфер.

Звуковые буферы

Звуковой буфер (sound buffer) содержит данные, которые находятся по пути к выводящему устройству. Возвратимся к нашей концепции ящика на ленточном конвейере: буфер — это ящик, переносящий ваш звук к динамикам. Если у вас нет достаточного количества звука для заполнения ящика, то он доставляется к динамикам не до конца заполненным, что может привести к интервалам в передаче аудио. Чем имеется у вас больше ящиков, тем больше звука вы можете заранее поместить в очередь, чтобы избежать интервалов (или замедлений). Обратной стороной всего этого является то, что для звука на конце динамика перехват звука, поставляемого приложением, занимает больше времени. Это может стать проблемой, если символ в вашей игре прыгает, но пользователь не слышит этого до тех пор, пока он не приземлится.

Когда звук готов к началу, создаются звуковые буферы и заполняются первыми фреймами звукового вывода вашего приложения. Минимальное количество необходимых буферов для начала воспроизведения на настольных системах Apple — всего лишь один, но на iPhone — это три. В приложениях, которые могут привести к интенсивному использованию процессора, более подходящим является использование большего количества буферов для предотвращения перегрузок. Чтобы подготовить буферы с первыми фреймами звуковых данных, заполняется каждый буфер в том порядке, в котором они создавались.

Это означает то, что к моменту заполнения буферов вам лучше уже иметь некоторое количество звука для их заполнения:

```
#define AUDIO_BUFFERS 3

unsigned long bufferSize;

bufferSize = aqc.frameCount * aqc.mDataFormat.mBytesPerFrame;
for (i=0; i<AUDIO_BUFFERS; i++) {
    AudioQueueAllocateBuffer(aqc.queue,
        bufferSize, &aqc.mBuffers[i]);
    AQBufferCallback (&aqc, aqc.queue, aqc.mBuffers[i]);
}
```

При выполнении этого кода аудиобуферы заполняются первыми фреймами звуковых данных из вашего приложения. Теперь очередь готова к активации, а это включает ленточный конвейер, отправляющий звуковые буферы к динамикам. Как только это произойдет, буферы освобождаются от своего содержимого (нет, память не обнуляется), а ящики возвращаются по ленточному конвейеру на повторное заполнение:

```
AudioQueueStart(aqc.queue, NULL);
```

Далее, когда вы готовы отключить звуковую очередь, просто воспользуйтесь функцией `AudioQueueDispose`, и все остановится:

```
AudioQueueDispose(aqc.queue, true);
```

Функция обратного вызова

Теперь аудиоочередь запущена, и каждую 1/60-ую часть секунды приложение запрашивается на заполнение данными нового звукового буфера. До сих пор еще не было объяснено, как это происходит. После опустошения буфера он готов к повторному заполнению, аудиоочередь вызывает функцию обратного вызова, заданную вами в качестве второго аргумента в `AudioQueueNewOutput`. Данная функция обратного вызова — это место, где приложение проделывает свою работу; она заполняет ящик, который переносит ваш выводимый звук к динамикам. Вы должны вызвать ее до начала очереди, чтобы заполнить звуковые буферы некоторым количеством начального звука. Затем очередь вызывает функцию каждый раз, когда буфер необходимо заполнить. При вызове вы заполните буфер аудиоочереди, передаваемый

путем копирования последнего звукового фрейма из вашего приложения — в нашем примере 735 дорожек:

```
static void AQBufferCallback(  
    void *aqc,  
    AudioQueueRef inQ,  
    AudioQueueBufferRef outQB)  
{
```

Структура обратного вызова, которую вы создали в самом начале, `aqc`, передается как определяемый пользователем аргумент, вместе с указателями на саму аудиоочередь и аудиобуфер для заполнения:

```
AQCallbackStruct *inData = (AQCallbackStruct *)aqc;
```

Поскольку структура `AQCallbackStruct` считается пользовательскими данными, то она передается в функцию обратного вызова в виде указателя на объект неизвестного типа (`void pointer`), и должна быть приведена к структуре `AQCallbackStruct` (здесь называемой `inData`) до того, как к ней может быть получен доступ. Данный код захватывает указатель на необработанные аудиоданные внутри буфера, в результате чего приложение может писать в него свой звук:

```
short *CoreAudioBuffer = (short *) outQB->mAudioData;
```

Переменная `CoreAudioBuffer` представляет пространство внутри звукового буфера, в который будут копироваться необработанные дорожки вашего приложения во время каждой синхронизации. Вашему приложению потребуется поддерживать тип "пишущая игла", чтобы отслеживать то, какой звук уже был отправлен в аудиоочередь:

```
if (inData->frameCount > 0) {
```

Переменная `frameCount` — это количество фреймов, которое ожидает увидеть буфер. Она должна равняться переменной `frameCount`, которая передавалась в структуру `AQCallbackStruct` (в нашем примере — 735):

```
outQB->mAudioDataByteSize = 4 * inData->frameCount;
```

Это то место, где вы точно сообщаете буферу, сколько данных он собирается принять: своего рода упаковочная ведомость для ящика. Общий размер выходного буфера должен равняться размеру обоих стереоканалов (два байта на канал = четыре байта), умноженному на количество отправленных фреймов (735):

```
for(i = 0 ; i < inData->frameCount * 2; i += 2) {
```

```
CoreAudioBuffer[i] = (LEFT CHANNEL DATA);  
CoreAudioBuffer[i+1] = (RIGHT CHANNEL DATA);  
}
```

Здесь функция обратного вызова пошагово проходит каждый выходной фрейм в буфере и копирует данные из того, что станет выводимым звуком вашего приложения в CoreAudioBuffer. Поскольку левый и правый каналы перемежаются, то циклу придется считаться с этим, пропуская в приращении по 2:

```
AudioQueueEnqueueBuffer(inQ, outQB, 0, NULL);  
} /* if (inData->frameCount > 0) */  
} /* AQBufferCallback */
```

Наконец, после того как фрейм будет скопирован в звуковой буфер, он помещается обратно в очередь воспроизведения.

Пример: проигрыватель PCM

Поскольку платформа Audio Toolbox проживает в стране C, то имеется неплохая возможность привести пример для iPhone, который не использует Objective-C или же платформу UIKit. В данном примере применяется старый добрый C, а сам пример запускается из командной строки с именем файла. Он загружает необработанный файл PCM, а затем воспроизводит его с помощью аудиоочереди Audio Toolbox. Поскольку ваше приложение, скорее всего, будет само порождать данные, и не использовать файл, то мы сначала будем читать файл в память буфера, а затем воспроизводить его из памяти буфера, чтобы проиллюстрировать практический принцип. Это должно послужить основанием для большинства приложений принять подобную архитектуру.

Поскольку необработанный файл PCM не содержит какую-либо информацию о частоте или размере фрейма, то данный пример вынужден будет присвоить свои собственные значения. Мы будем использовать формат для 16-битных 44 кГц несжатых моноданных PCM. Это задается тремя определениями, сделанными в самом верхе программы:

```
#define BYTES_PER_SAMPLE 2  
16-bit = 2 bytes  
#define SAMPLE_RATE 44100
```

```
44,100 samples per second = 44 KHz
typedef unsigned short sampleFrame;
```

`unsigned short` равен двум байтам (на дорожку).

Если вы не можете найти необработанный файл PCM для выполнения этого примера, то можете использовать wav-файл при условии, что он закодирован в 16-битном 44-килогерцовом необработанном формате PCM. Иначе вы можете адаптировать этот пример для использования другой кодировки, изменив `mFormatID` внутри структуры аудиоочереди. Этот пример не будет пытаться произвести разбор заголовков wav-файла, он всего лишь предполагает, что предоставляемые вами данные являются необработанными, т. е. такими, какие предоставляют игры или приложения другого типа. Заголовки wav-файла будут передаваться в аудиоканал с остальными данными, поэтому, возможно, вы будете слышать легкий щелчок или хлопок перед началом воспроизведения необработанных звуковых данных внутри файла.

Чтобы скомпилировать данный пример с помощью пакета инструментов, воспользуйтесь командной строкой:

```
$ arm-apple-darwin9-gcc -o playpcm playpcm.c
  -framework AudioToolbox -framework CoreAudio
  -framework CoreFoundation
  -DMAC_OS_X_VERSION_MAX_ALLOWED=MAC_OS_X_VERSION_10_5
  -DMAC_OS_X_VERSION_MIN_REQUIRED=MAC_OS_X_VERSION_10_5
```

Поскольку Leopard также включает в себя платформу Audio Toolbox, то данный пример может быть скомпилирован и на настольной системе:

```
$ gcc -o playpcm playpcm.c
  -framework AudioToolbox -framework CoreAudio -framework CoreFoundation
```

В листинге 6.3 приведен сам код.

Листинг 6.3. Пример Audio Toolbox (playpcm.c)

```
#include <stdio.h>
#include <stdlib.h>
#include <errno.h>
#include <sys/stat.h>
#include <AudioToolbox/AudioQueue.h>
```

```
#define BYTES_PER_SAMPLE 2
#define SAMPLE_RATE 44100
typedef unsigned short sampleFrame;

#define FRAME_COUNT 735
#define AUDIO_BUFFERS 3

typedef struct AQCallbackStruct {
    AudioQueueRef queue;
    UInt32 frameCount;
    AudioQueueBufferRef mBuffers[AUDIO_BUFFERS];
    AudioStreamBasicDescription mDataFormat;
    UInt32 playPtr;
    UInt32 sampleLen;
    sampleFrame *pcmBuffer;
} AQCallbackStruct;

void *loadpcm(const char *filename, unsigned long *len);
int playbuffer(void *pcm, unsigned long len);
void AQBufferCallback(void *in, AudioQueueRef inQ,
    AudioQueueBufferRef outQB);

int main(int argc, char *argv[]) {
    char *filename;
    unsigned long len;
    void *pcmbuffer;
    int ret;

    if (argc < 2) {
        fprintf(stderr, "Syntax: %s [filename]\n", argv[0]);
        exit(EXIT_FAILURE);
    }

    filename = argv[1];
    pcmbuffer = loadpcm(filename, &len);
```

```
if (!pcmbuffer) {
    fprintf(stderr, "%s: %s\n", filename, strerror(errno));
    exit(EXIT_FAILURE);
}

ret = playbuffer(pcmbuffer, len);
free(pcmbuffer);
return ret;
}

void *loadpcm(const char *filename, unsigned long *len) {
    FILE *file;
    struct stat s;
    void *pcm;

    if (stat(filename, &s))
        return NULL;
    *len = s.st_size;
    pcm = (void *) malloc(s.st_size);
    if (!pcm)
        return NULL;
    file = fopen(filename, "rb");
    if (!file) {
        free(pcm);
        return NULL;
    }
    fread(pcm, s.st_size, 1, file);
    fclose(file);
    return pcm;
}

int playbuffer(void *pcmbuffer, unsigned long len) {
    AQCallbackStruct aqc;
    UInt32 err, bufferSize;
    int i;
```

```
aqc.mDataFormat.mSampleRate = SAMPLE_RATE;
aqc.mDataFormat.mFormatID = kAudioFormatLinearPCM;
aqc.mDataFormat.mFormatFlags = kLinearPCMFormatFlagIsSignedInteger
    | kAudioFormatFlagIsPacked;
aqc.mDataFormat.mBytesPerPacket = 4;
aqc.mDataFormat.mFramesPerPacket = 1;
aqc.mDataFormat.mBytesPerFrame = 4;
aqc.mDataFormat.mChannelsPerFrame = 2;
aqc.mDataFormat.mBitsPerChannel = 16;
aqc.frameCount = FRAME_COUNT;
aqc.sampleLen = len / BYTES_PER_SAMPLE;
aqc.playPtr = 0;
aqc.pcmBuffer = pcmbuffer;

err = AudioQueueNewOutput(&aqc.mDataFormat, AQBufferCallback,
    &aqc, NULL, kCFRunLoopCommonModes,
    0, &aqc.queue);

if (err)
    return err;

aqc.frameCount = FRAME_COUNT;
bufferSize = aqc.frameCount * aqc.mDataFormat.mBytesPerFrame;
for (i=0; i<AUDIO_BUFFERS; i++) {
    err = AudioQueueAllocateBuffer(aqc.queue, bufferSize,
        &aqc.mBuffers[i]);
    if (err)
        return err;
    AQBufferCallback(&aqc, aqc.queue, aqc.mBuffers[i]);
}

err = AudioQueueStart(aqc.queue, NULL);
if (err)
    return err;

while(aqc.playPtr < aqc.sampleLen)
    { select(NULL, NULL, NULL, NULL, 1.0); }
```

```
    sleep(1);
    return 0;
}

void AQBufferCallback(void *in, AudioQueueRef inQ,
                     AudioQueueBufferRef outQB)
{
    AQCallbackStruct *aqc;
    short *coreAudioBuffer;
    short sample;
    int i;

    aqc = (AQCallbackStruct *) in;
    coreAudioBuffer = (short*) outQB->mAudioData;

    printf("Sync: %ld / %ld\n", aqc->playPtr, aqc->sampleLen);
    if (aqc->playPtr >= aqc->sampleLen) {
        AudioQueueDispose(aqc->queue, true);
        return;
    }

    if (aqc->frameCount > 0) {
        outQB->mAudioDataByteSize = 4 * aqc->frameCount;
        for(i=0; i<aqc->frameCount*2; i+=2) {
            if (aqc->playPtr > aqc->sampleLen || aqc->playPtr < 0)
                sample = 0;
            else
                sample = (aqc->pcmBuffer[aqc->playPtr]);
            coreAudioBuffer[i] = sample;
            coreAudioBuffer[i+1] = sample;
            aqc->playPtr++;
        }
        AudioQueueEnqueueBuffer(inQ, outQB, 0, NULL);
    }
}
```

Как это работает

Вот как работает программа `playpcm`:

1. В начале программы вызывается функция приложения `main()`, которая извлекает имя файла из списка аргументов (передаваемого в командной строке).
2. Функция `main()` вызывает `loadpcm()`, которая определяет длину аудиофайла и загружает его в память, возвращая этот буфер функции `main()`.
3. Вызывается функция `playbuffer()` с содержимым этой памяти и ее длиной. Эта функция создает нашу определяемую пользователем структуру `AQCallbackStruct`, чья конструкция декларируется в начале программы. Эта структура хранит указатели на аудиоочередь, звуковые буферы и память, хранящую содержимое загруженного файла. Кроме того, она содержит длину дорожки и целое число `playPtr`, которое выступает в роли записывающей иглы, определяя последнюю дорожку, скопированную в звуковой буфер.
4. Инициализируется и запускается новая звуковая очередь. Чтобы синхронизировать первые дорожки в памяти, вызывается функция обратного вызова по одному разу для каждого используемого звукового буфера. Затем стартует аудиоочередь. После этого программа сидит и ждет, когда закончится воспроизведение текущей дорожки.
5. По мере воспроизведения аудио один за другим опустошаются звуковые буферы. Каждый раз, когда какому-либо буферу необходимы еще звуковые данные, вызывается функция `AQBufferCallback`.
6. Функция `AQBufferCallback` увеличивает `playPtr` и копирует следующие звуковые фреймы из памяти для воспроизведения в звуковой буфер. Поскольку необработанные дорожки PCM являются моно, то одинаковые данные копируются как в левый, так и в правый выходной канал.
7. Когда `playPtr` превысит длину звуковой дорожки, то цикл ожидания в `playpcm()` разорвется, в результате чего функция вернется обратно к `main()` для освобождения ресурсов и выхода.

Для дальнейшего изучения

Постарайтесь выполнить следующие задания.

- Модифицируйте этот пример для воспроизведения 8-битного звука PCM, изменив тип данных `sampleFrame` и `BYTES_PER_SAMPLE`. Также вам при-

дется увеличить уровень громкости, поскольку звуковая дорожка теперь имеет длину в один байт, а канал аудиоочереди имеет длину в два байта.

- Проверьте наличие AudioQueue.h в Mac OS X Leopard на настольной системе. Вы сможете найти его в папке /System/Library/Frameworks/AudioToolbox.framework/Headers/.

Запись звука

Запись звука происходит совершенно аналогичным образом, как и его воспроизведение; однако создаваемая очередь задается как очередь записи, храня в себе ввод, а не вывод. Звук может быть записан во множестве различных форматов, включая Apple Lossless, PCM и др. Приводимый в этом разделе пример проводит четкую параллель с нашим предыдущим примером аудиоочереди, но с некоторыми изменениями. Мы прокомментируем их в самом примере. При записи звука ленточный конвейер аудиоочереди вращается в обратную сторону. Микрофон iPhone делает всю работу по заполнению ящиков звуком и отправки их от микрофона к вашему приложению. Вы остаетесь ответственными за информирование платформы о том, какой формат и частоту дорожки вы бы хотели, но теперь вы отвечаете не за заполнение ящиков, а за их опустошение и запись на диск. Для записи непосредственно в файл, а не копирования его в память, вы будете использовать функции аудиофайлов из Audio Toolbox. Очередь записи строго придерживается принципа "первым прибыл — первым обслужен" (first in, first out), т. е. ленточный конвейер воспроизводит дорожки именно в том порядке, в каком они были записаны.

Очередь записи Audio Toolbox работает следующим образом:

1. Создается аудиоочередь и задаются свойства, определяющие тип звука, который будет записываться (формат, частота дорожки и т. д.).
2. К очереди прикрепляются звуковые буферы, которые будут содержать сами звуковые фреймы по мере их записи. Звуковой фрейм можно рассматривать как отдельный ящик, заполненный записанным звуком, в то время как дорожка — это отдельная часть цифрового звука внутри фрейма.
3. Разработчик предоставляет "входную" функцию обратного вызова, которая вызывает аудиоочередь каждый раз, когда звуковой буфер заполняется записанным аудио. Эта функция обратного вызова отвечает за запись записанных фреймов на диск, а затем отправку ящика по кругу для следующего заполнения.

Структура аудиоочереди

Как мы уже объясняли ранее, платформа Audio Toolbox использует низкоуровневые интерфейсы C, поэтому в ней нет понятия класса. Структура обратного вызова изначально создается для того, чтобы содержать все переменные, которые будут перемещаться по вашему записывающему приложению. Структура `AQCallbackStruct`, приведенная далее, похожа на свою версию обратного, но с несколькими добавлениями:

```
typedef struct AQCallbackStruct {  
    AudioStreamBasicDescription mDataFormat;  
    AudioQueueRef queue;  
    AudioQueueBufferRef mBuffers[AUDIO_BUFFERS];  
    AudioFileID outputFile;  
    unsigned long frameSize;  
    long long recPtr;  
    int run;  
} AQCallbackStruct;
```

Следующие компоненты сгруппированы в данную структуру для обслуживания аудиоплатформы:

- `AudioQueueRef queue` — указатель на объект аудиоочереди, который создаст ваша программа;
- `AudioFileID outputFile` — указатель на выходной файл, который будет записан после того, как запишется звук;
- `unsigned long frameSize` — общее количество дорожек для копирования за одну аудиосинхронизацию. Это полностью отдается на откуп реализующему;
- `long long recPtr` — числовой указатель на текущую позицию записывающей "иглой" на основе того, что приложение уже обработало. Он увеличивается по мере увеличения записываемых данных;
- `int run` — значение этой переменной определяет необходимость повторной постановки в очередь звуковых буферов, т. е. надо или нет отправлять ящики назад для повторного заполнения. Когда придет время завершить запись, это значение должно быть установлено вашим приложением в нулевое значение;
- `AudioQueueBufferRef mBuffers` — массив, содержащий общее количество звуковых буферов, которые будут использоваться. Корректное количе-

ство элементов будет обсуждаться в разд. "Звуковые буферы" далее в этой главе;

- `AudioStreamBasicDescription mDataFormat` — информация о формате аудио, которое будет записываться.

Прежде чем создать очередь записи, необходимо инициализировать экземпляры всех этих переменных:

```
AQCallbackStruct aqc;
aqc.mDataFormat.mFormatID = kAudioFormatLinearPCM;
aqc.mDataFormat.mSampleRate = 44100.0;
aqc.mDataFormat.mChannelsPerFrame = 2;
aqc.mDataFormat.mBitsPerChannel = 16;
aqc.mDataFormat.mBytesPerPacket =
aqc.mDataFormat.mBytesPerFrame =
aqc.mDataFormat.mChannelsPerFrame * sizeof (short int);
aqc.mDataFormat.mFramesPerPacket = 1;
aqc.mDataFormat.mFormatFlags = kLinearPCMFormatFlagIsBigEndian
                               | kLinearPCMFormatFlagIsSignedInteger
                               | kLinearPCMFormatFlagIsPacked;
aqc.frameSize = 735;
```

В данном примере мы подготовили структуру для 16-битного стереозвука (два канала) с частотой дорожки 44 кГц (44 100). Выходная дорожка будет представлена в виде двух двухбайтных *коротких* (short) целых чисел, следовательно, по четыре байта на фрейм (по два байта на левый и правый каналы).

Частота дорожки и размер фрейма предписывают то, как часто ваше приложение будет получать звук. При частоте 44 100 дорожек в секунду приложение можно заставить синхронизировать звук каждую 1/60-ую часть секунды, задав размер фрейма в 735 дорожек ($44100/60 = 735$). Это очень высокая интенсивность синхронизации для приложений, обрабатывающих звук в реальном времени, поэтому если вам не нужна такая частая синхронизация, то вы можете выбрать больший размер фрейма, например, 22 050, который будет синхронизировать каждые полсекунды.

В данном примере будет использоваться формат PCM (необработанные данные), но в iPhone поддерживаются и другие форматы:

- `kAudioFormatLinearPCM`;
- `kAudioFormatAppleIMA4`;

- kAudioFormatMPEG4AAC;
- kAudioFormatULaw;
- kAudioFormatALaw;
- kAudioFormatMPEGLayer3;
- kAudioFormatAppleLossless;
- kAudioFormatAMR.

Подготовка аудиовывода

После того как свойства аудиоочереди будут определены, можно подготовить объект новой аудиоочереди. За подготовку входного (записывающего) канала и прикрепление его к очереди отвечает функция `AudioQueueNewInput`. Прототип функции выглядит следующим образом:

```
AudioQueueNewOutput (
    const AudioStreamBasicDescription *inFormat,
    AudioQueueOutputCallback inCallbackProc,
    void * inUserData,
    CFRunLoopRef inCallbackRunLoop,
    CFStringRef inCallbackRunLoopMode,
    UInt32 inFlags,
    AudioQueueRef * outAQ);
```

Здесь:

- `inFormat` — указатель на структуру, описывающую аудиоформат, который будет воспроизводиться. Мы определили эту структуру ранее как члена типа данных `AudioStreamBasicDescription` в рамках нашей структуры `AQCallbackStruct`;
- `inCallbackProc` — имя функции обратного вызова, которая будет вызываться, когда буфер аудиоочереди будет заполнен записанными данными;
- `inUserData` — указатель на данные, которые разработчик при желании может передавать в функцию обратного вызова. Он будет содержать указатель на экземпляр определяемой пользователем структуры `AQCallbackStruct`, которая должна содержать информацию как об аудиоочереди, так и любую другую относящуюся к приложению информацию о записываемых дорожках;

- `inCallbackRunLoopMode` — сообщает аудиоочереди о том, какое должно быть заикливание аудио. Если указан `NULL`, то функция обратного вызова вызывается каждый раз, когда опустошается звуковой буфер. Для запуска обратного вызова при других условиях существуют дополнительные модели;
- `inFlags` — не используется, зарезервировано;
- `outAO` — когда функция `AudioQueueNewOutput` возвращает, данный указатель устанавливается на только что созданную аудиоочередь. Присутствие данного аргумента позволяет коду ошибки использоваться в качестве возвращаемого значения данной функции.

Действительный вызов данной функции с использованием созданной ранее структуры аудиоочереди выглядит следующим образом:

```
AudioQueueNewInput (&aqc.mDataFormat,  
                  AQInputCallback,  
                  &aqc,  
                  NULL,  
                  kCFRunLoopCommonModes,  
                  0,  
                  &aqc.queue);
```

В данном примере имя функции обратного вызова определено как `AQBufferCallback`. Эта функция будет создана в нескольких следующих разделах. Это функция, которая будет отвечать за прием записываемого звука и копирование его на диск.

Звуковые буферы

Звуковой буфер (sound buffer) содержит данные, которые находятся по пути к выводящему устройству. Возвращаясь к нашей концепции ящика на ленточном конвейере, можно сказать, что буфер — это ящик, переносящий ваш звук между микрофоном и вашей функцией обратного вызова. Если iPhone не может обеспечить достаточное количество звука в канале, то это может привести к интервалам в вашей записи. Чем больше ящиков у вас имеется, тем больше звука вы можете заранее поместить в очередь, чтобы избежать интервалов (или замедлений). Обратной стороной всего этого является то, что для звука на конце микрофона перехват звука, поставляемого приложением, занимает больше времени. Это может стать проблемой, если вы пишете син-

тезатор речи или приложение другого типа, требующего максимального приближения к звуку реального времени. Когда звук готов к началу, создаются звуковые буферы и заполняются первыми фреймами микрофонного ввода. Минимальное количество необходимых буферов для начала воспроизведения на настольных системах Apple — всего лишь один, но на iPhone — это три. В приложениях, которые могут привести к интенсивному использованию процессора, более подходящим является использование большего количества буферов для предотвращения перегрузок. Чтобы подготовить буферы с первыми фреймами звуковых данных, заполняется каждый буфер в том порядке, в котором они создавались. Это означает то, что к моменту заполнения буферов вам лучше уже иметь некоторое количество звука для их заполнения:

```
#define AUDIO_BUFFERS 3

for (i=0; i<AUDIO_BUFFERS; i++) {
    AudioQueueAllocateBuffer (aqc.queue, aqc.frameSize, &aqc.mBuffers[i]);
    AudioQueueEnqueueBuffer (aqc.queue, aqc.mBuffers[i], 0, NULL);
}
```

В примере воспроизведения аудиобуферы отправляются к функции обратного вызова для заполнения данными. Поскольку данный пример демонстрирует запись звука, а не его воспроизведение, то буфер должен быть сначала поставлен в очередь (отправлен обратно по ленточному конвейеру), чтобы аудиоокружение могло заполнить его записанными данными. После заполнения окружение автоматически вызовет вашу функцию `AQInputCallback`, которая выгрузит его содержимое на диск и отправит обратно за следующей порцией данных. При выполнении данного кода аудиобуферы заполняются первыми фреймами звуковых данных с микрофона и отправляются к функции обратного вызова. Теперь очередь готова к активации, что приводит к тому, что ленточный конвейер отправляет звуковые буферы от микрофона к вам. Как только это произойдет, буферы освобождаются от своего содержимого (нет, память не обнуляется), а ящики возвращаются по ленточному конвейеру на повторное заполнение:

```
AudioQueueStart (aqc.queue, NULL);
```

Далее, когда вы готовы отключить запись звука, просто воспользуйтесь функцией `AudioQueueDispose`, и все остановится:

```
AudioQueueStop (aqc.queue, true);
```

```
AudioQueueDispose (aqc.queue, true);
```

Функция обратного вызова

Теперь аудиоочередь запущена и каждую 1/60-ую часть секунды приложению предоставляется звуковой буфер, содержащий записанный фрейм. До сих пор еще не было объяснено, как это происходит. После того как буфер будет заполнен и готов к опустошению, аудиоочередь вызывает функцию обратного вызова, заданную вами в качестве второго аргумента в `AudioQueueNewInput`. Данная функция обратного вызова — это место, где приложение проделывает свою работу; она опустошает ящик, который переносит ваше записанное аудио, записывает содержимое на диск или сохраняет его как-либо иначе. Очередь вызывает функцию каждый раз, когда буфер необходимо опустошить. При вызове вы записываете буфер аудиоочереди — в нашем примере 735 дорожек:

```
static void AQInputCallback (
    void *aqr,
    AudioQueueRef inQ,
    AudioQueueBufferRef inQB,
    const AudioTimeStamp *timestamp,
    unsigned long frameSize,
    const AudioStreamPacketDescription *mDataFormat)
{
```

Структура обратного вызова `aqc`, которую вы создали в самом начале, передается как определяемый пользователем аргумент, вместе с указателями на саму аудиоочередь и аудиобуфер для опустошения:

```
AQCallbackStruct *aqc = (AQCallbackStruct *)aqr;
```

Поскольку структура `AQCallbackStruct` считается пользовательскими данными, то она передается в функцию обратного вызова в виде указателя на объект неизвестного типа (`void pointer`), и должна быть приведена к структуре `AQCallbackStruct` (здесь называемой `inData`) до того, как к ней может быть получен доступ. Данный код захватывает указатель на необработанные аудиоданные внутри буфера, в результате чего приложение может читать из него звук:

```
short *CoreAudioBuffer = (short *) inQB->mAudioData;
```

Переменная `CoreAudioBuffer` представляет пространство внутри звукового буфера, в который будут копироваться необработанные записанные дорожки

вашего приложения во время каждой синхронизации. Вашему приложению потребуется поддерживать тип "пишущая игла", чтобы отслеживать то, какой звук уже был считан из аудиоочереди:

```
int recNeedle = 0;
myBuffer = malloc(aqc.frameSize * nSamples);
...
static void AQInputCallback (
    void *aqr,
    AudioQueueRef inQ,
    AudioQueueBufferRef inQB,
    const AudioTimeStamp *timestamp,
    unsigned long frameSize,
    const AudioStreamPacketDescription *mDataFormat)
{
    AQCallbackStruct *aqc = (AQCallbackStruct *) aqr;

    short *CoreAudioBuffer = (short *) inQB->mAudioData;
    memcpy(myBuffer + recNeedle, CoreAudioBuffer,
        aqc.mDataFormat.mBytesPerFrame * aqc.frameSize);
    recNeedle += aqc.frameSize;
    if (!aqc->run)
        return;

    AudioQueueEnqueueBuffer (aqc->queue, inQB, 0, NULL);
}
```

Чтобы писать в файл, вы будете использовать набор функций `AudioFile` из `Audio Toolbox`. Чтобы подготовить аудиофайл, сначала вам потребуется определить формат файла. Приведенный далее код определяет свойство, необходимое для аудиофайла `AIFF`:

```
AudioFileTypeID fileFormat = kAudioFileAIFFFormat;
```

Для хранения действительного пути к аудиофайлу вы будете использовать структуру `CFURL`:

```
CFURLRef filename = CFURLCreateFromFileSystemRepresentation(NULL,
    (const unsigned char *) path_to_file,
    strlen (path_to_file),
    false);
```

Наконец, создается сам аудиофайл с помощью вызова функции `AudioFileCreateWithURL`, содержащей имя файла и только что созданных свойств формата. Указатель на файл записывается в структуру `AQCallbackStruct`, так что теперь мы знаем, как получить доступ к данному файлу, когда нам понадобится писать в него звук:

```
AudioFileCreateWithURL(filename, fileFormat, &aqc.mDataFormat,  
                      kAudioFileFlags_EraseFile, &aqc.mAudioFile);
```

По мере записи новых аудиодорожек вы будете писать в этот файл с помощью функции `AudioFileWritePackets`, которая является еще одной функцией, встроенной в `Audio Toolbox` специально для записи аудиопакетов в файл. В приведенном далее коде будет показан пример ее использования.

Пример: магнитофон

Данный пример может запускаться из командной строки с именем файла и длительностью как на iPhone, так и на настольной системе. Он записывает данные с микрофона в течение заданного периода времени и сохраняет их в файл с указанным именем. Поскольку `Leonard` также содержит платформу `Audio Toolbox`, то данный пример может быть скомпилирован как для iPhone, так и для настольной системы:

```
$ arm-apple-darwin9-gcc -o recorder recorder.c 🐣  
  -framework AudioToolbox -framework CoreAudio 🐣  
  -framework CoreFoundation 🐣  
  -DMAC_OS_X_VERSION_MAX_ALLOWED=MAC_OS_X_VERSION_10_5 🐣  
  -DMAC_OS_X_VERSION_MIN_REQUIRED=MAC_OS_X_VERSION_10_5  
  
$ gcc -o recorder recorder.c 🐣  
  -framework AudioToolbox -framework CoreAudio -framework CoreFoundation
```

В листинге 6.4 приведен соответствующий код.

Листинг 6.4. Пример звукозаписи с использованием `Audio Toolbox` (`recorder.c`)

```
#include <AudioToolbox/AudioQueue.h>  
#include <AudioToolbox/AudioFile.h>  
#include <AudioToolbox/AudioConverter.h>
```

```
#include <stdio.h>
#include <stdlib.h>
#include <errno.h>
#include <sys/stat.h>
#include <sys/select.h>
#define AUDIO_BUFFERS 3

typedef struct AQCallbackStruct {
    AudioStreamBasicDescription mDataFormat;
    AudioQueueRef queue;
    AudioQueueBufferRef mBuffers[AUDIO_BUFFERS];
    AudioFileID outputFile;
    unsigned long frameSize;
    long long recPtr;
    int run;
} AQCallbackStruct;

static void AQInputCallback (
    void *aqr,
    AudioQueueRef inQ,
    AudioQueueBufferRef inQB,
    const AudioTimeStamp *timestamp,
    unsigned long frameSize,
    const AudioStreamPacketDescription *mDataFormat)
{
    AQCallbackStruct *aqc = (AQCallbackStruct *) aqr;

    /* Записываем данные в файл */
    if (AudioFileWritePackets (aqc->outputFile, false,
        inQB->mAudioDataByteSize,
        mDataFormat, aqc->recPtr, &frameSize, inQB->mAudioData) == noErr)
    {
        aqc->recPtr += frameSize;
    }
}
```

```
/* Если мы собираемся остановить запись, то не ставим
   звуковые буферы в очередь. */
if (!aqc->run)
    return;

AudioQueueEnqueueBuffer (aqc->queue, inQB, 0, NULL);
}

int main(int argc, char *argv[]) {
    AQCallbackStruct aqc;
    AudioFileTypeID fileFormat;
    CFURLRef filename;
    struct timeval tv;
    int i;

    if (argc < 3) {
        fprintf(stderr, "Syntax: %s [filename] [duration]", argv[0]);
        exit(EXIT_FAILURE);
    }

    aqc.mDataFormat.mFormatID = kAudioFormatLinearPCM;
    aqc.mDataFormat.mSampleRate = 44100.0;
    aqc.mDataFormat.mChannelsPerFrame = 2;
    aqc.mDataFormat.mBitsPerChannel = 16;
    aqc.mDataFormat.mBytesPerPacket =
    aqc.mDataFormat.mBytesPerFrame =
    aqc.mDataFormat.mChannelsPerFrame * sizeof (short int);
    aqc.mDataFormat.mFramesPerPacket = 1;
    aqc.mDataFormat.mFormatFlags = kLinearPCMFormatFlagIsBigEndian
        | kLinearPCMFormatFlagIsSignedInteger
        | kLinearPCMFormatFlagIsPacked;
    aqc.frameSize = 735;

    AudioQueueNewInput (&aqc.mDataFormat, AQInputCallback, &aqc, NULL,
        kCFRunLoopCommonModes, 0, &aqc.queue);
}
```

```
/* Создаем выходной файл */

fileFormat = kAudioFileAIFFFormat;
filename = CFURLCreateFromFileSystemRepresentation (NULL, argv[1],
    strlen (argv[1]), false);

AudioFileCreateWithURL(filename, fileFormat,
    &aqc.mDataFormat, kAudioFileFlags_EraseFile,
    &aqc.outputFile);

/* Инициализируем буферы записи */
for (i=0; i<AUDIO_BUFFERS; i++) {
    AudioQueueAllocateBuffer (aqc.queue, aqc.frameSize,
        &aqc.mBuffers[i]);
    AudioQueueEnqueueBuffer (aqc.queue, aqc.mBuffers[i], 0, NULL);
}

aqc.recPtr = 0;
aqc.run = 1;

AudioQueueStart (aqc.queue, NULL);

/* Ждем некоторое время, пока идет запись */
tv.tv_sec = atof(argv[2]);
tv.tv_usec = 0;
select(0, NULL, NULL, NULL, &tv);

/* Завершаем запись */
AudioQueueStop (aqc.queue, true);
aqc.run = 0;

AudioQueueDispose (aqc.queue, true);
AudioFileClose (aqc.outputFile);

exit(EXIT_SUCCESS);
}
```

Как это работает

Программа `record` работает следующим образом:

1. В начале программы вызывается функция приложения `main()`, которая извлекает имя файла из списка аргументов (передаваемого в командной строке).
2. Функция `main()` создает нашу задаваемую пользователем структуру, чья конструкция декларируется в начале программы. Эта структура хранит указатели на аудиоочередь, звуковые буферы и созданный выходной файл. Кроме того, она содержит длину дорожки и целое число `recPtr`, которое выступает в роли записывающей иглы, определяя последнюю дорожку, записанную на диск.
3. Инициализируется и запускается новая звуковая очередь. Инициализируется и ставится в эту очередь записи каждый звуковой буфер. Затем стартует аудиоочередь. После этого программа ждет, когда закончится запись текущей дорожки.
4. По мере записи аудио звуковые буферы отправляются к функции обратного вызова, где они, один за другим, опустошаются. Каждый раз, когда какой-либо буфер опустошается, вызывается функция `AQInputCallback`.
5. Функция `AQInputCallback` увеличивает значение записывающей иглы и копирует звуковой фрейм на диск.

Уровень громкости

Дорожки, воспроизводимые посредством платформы `Celestial`, проигрываются на достаточно высоком уровне громкости, чтобы автоматически отслеживать его с помощью системного уровня громкости. Однако низкоуровневая платформа, каковой является `Audio Toolbox`, не обращает внимания на системный уровень громкости, поэтому выходной уровень громкости является статическим и независимым от установленного на iPhone. Управление уровнем громкости аудиоочереди требует от разработчика использования функций более высокого уровня для чтения уровня громкости и масштабирования звукового потока для согласования с ним.

`Audio Toolbox` и `Celestial` встречаются, когда вы работаете с настройками высокого уровня, например, уровнем громкости. Уровень громкости звука является функцией `mediaserverd` — аудиодемона, с которым вы познакомились ранее при изучении `Core Audio`. Данный демон крепко-накрепко связан с

платформой Celestial. Поэтому платформа Celestial может быть использована для чтения уровня громкости и перехвата нажатий кнопок регулировки уровня громкости.

Прежде чем приступить к описанию уровня громкости, важно обсудить, что делать с ним при воспроизведении посредством Audio Toolbox. Уровень громкости передается в виде значения между 0,0 и 1,0 (т. е. 0% и 100%). В функции обратного вызова, использованной в предыдущем разделе, звуковые фреймы копировались из вывода приложения в звуковые буферы при каждой синхронизации. Мы предполагаем, что приложение получило текущий уровень громкости и сохранило его в переменной `_volume`, выделенный полужирным начертанием текст в следующем примере показывает изменения, которые необходимо сделать в данной процедуре, чтобы встроить возможность настройки уровня громкости пользователем:

```
for(i=0; i<aqc->frameCount*2; i+=2) {
    if (aqc->playPtr > aqc->sampleLen || aqc->playPtr < 0)
        sample = 0;
    else
        sample = (aqc->pcmBuffer[aqc->playPtr]);
    coreAudioBuffer[i] = sample * _volume;
    coreAudioBuffer[i+1] = sample * _volume;
    aqc->playPtr++;
}
```

Другими словами, если уровень громкости установлен на свой максимум, то воспроизводятся текущие данные дорожки (т. е. `sample * 1.0`). Если же уровень громкости установлен в любое другое значение, то уровень громкости дорожки умножается на значение уровня громкости, так что он будет уменьшен на соответствующий коэффициент уровня громкости. Если вы хотите, чтобы максимальный уровень громкости был еще громче, то можете просто умножить `_volume` на коэффициент 2 или 3, хотя тем самым вы идете на риск износа вашего аудио.

Чтение уровня громкости

Audio Toolbox "живет в стране" C, в то время как Celestial требует контекста Objective-C. Чтобы проще всего объединить эти два мира и позволить им обмениваться данными, надо использовать глобальную переменную. В данном примере такая глобальная переменная называется `_volume`.

Celestial делегирует управление уровнем громкости и звонком классу AVSystemController. Чтобы прочитать уровень громкости из Celestial, создайте его экземпляр:

```
NSString *audioDeviceName;
float _volume;
AVSystemController *avs =
    [ AVSystemController sharedAVSystemController ];
[ avs getActiveCategoryVolume: &_volume andName: &audioDeviceName ];
```

Когда метод `getActiveCategoryVolume` уведомлен, он устанавливает в качестве значения `_volume` текущий уровень громкости в диапазоне от 0,0 до 1,0. Это автоматически будет распознано кодом Audio Toolbox, при условии, что `_volume` является глобальной переменной, и приведет к тому, что все следующие фреймы дорожек будут умножаться на новое значение.

Уведомления об изменении уровня громкости

Использование метода `getActiveCategoryVolume` в качестве одноразового чтения полезно для установки выходного уровня громкости при старте программы, но он не будет менять что-либо, если уровень громкости меняется в процессе использования приложения. Чтобы это сделать, добавьте в приложение *наблюдателя* (observer). Этот наблюдатель будет отслеживать специальные системные события и уведомлять данный метод, когда такое событие будет происходить:

```
[ [ NotificationCenter defaultCenter ] addObserver: self
    selector:@selector(volumeChange:)
    name: @"AVSystemController_SystemVolumeDidChangeNotification"
    object: avs ];
```

Данный код задает метод `volumeChange` в качестве наблюдателя за изменениями системного уровня громкости. Затем метод `volumeChange` определяется в вызывающем классе:

```
- (void)volumeChange:(NSNotification *)notification {
    AVSystemController *avsc = [ notification object ];
    NSString *audioDeviceName;
    [ avsc getActiveCategoryVolume:&_volume
        andName:&audioDeviceName ];
}
```

Когда метод `volumeChange` уведомляется, в него передается объект `AVSystemController` вместе с уведомлением. Затем данный объект используется для повторного чтения уровня громкости в `_volume`, где он будет подобран функцией `AQBufferCallback`, заполняющей аудиоочередь.

Пример: какой у меня уровень громкости?

Данный пример аналогичен примеру "Hello, World!" из главы 3 за исключением того, что отображается уровень громкости, а не приветственное сообщение. Когда нажимается одна из кнопок регулировки уровня громкости, созданный нами наблюдатель уведомляет метод `volumeChanged`, который перепроверяет уровень громкости и обновляет текст.

Данный пример может быть скомпилирован с помощью приведенной далее командной строки. Вам придется подключить платформы `Celestial`, `Core Audio` и `Audio Toolbox`:

```
$ arm-apple-darwin9-gcc -o MyExample MyExample.m -lobjc \
    -framework CoreFoundation -framework Foundation -framework UIKit \
    -framework Celestial -framework AudioToolbox -framework CoreAudio \
    -DMAC_OS_X_VERSION_MAX_ALLOWED=MAC_OS_X_VERSION_10_5 \
    -DMAC_OS_X_VERSION_MIN_REQUIRED=MAC_OS_X_VERSION_10_5
```

В листингах 6.5 и 6.6 приведен соответствующий код.

Листинг 6.5. Пример управления уровнем громкости (MyExample.h)

```
#import <CoreFoundation/CoreFoundation.h>
#import <UIKit/UIKit.h>
#import <UIKit/UITextView.h>
#import <Celestial/AVSystemController.h>

@interface MainView : UIView
{
    UITextView *textView;
    AVSystemController *avs;
}
```

```
- (id)initWithFrame:(CGRect)frame;
- (void)dealloc;
- (void)displayVolume;

@end

@interface MyApp : UIApplication
{
    UIWindow *window;
    MainView *mainView;
}

- (void)applicationDidFinishLaunching:(NSNotification *)aNotification;
@end
```

Листинг 6.6. Пример управления уровнем громкости (MyExample.m)

```
#import "MyExample.h"

float _volume;

int main(int argc, char **argv)
{
    NSAutoreleasePool *autoreleasePool = [
        [NSAutoreleasePool alloc] init
    ];
    int returnCode = UIApplicationMain(argc, argv, @"MyApp", @"MyApp");
    [autoreleasePool release];
    return returnCode;
}

@implementation MyApp

- (void)applicationDidFinishLaunching:(NSNotification *)aNotification {
    window = [ [UIWindow alloc] initWithContentRect:
```

```
[ UIHardware fullScreenApplicationContentRect ]
];

CGRect rect = [ UIHardware fullScreenApplicationContentRect ];
rect.origin.x = rect.origin.y = 0.0f;

mainView = [ [ MainView alloc ] initWithFrame: rect ];

[ window setContentView: mainView ];
[ window orderFront: self ];
[ window makeKey: self ];
[ window _setHidden: NO ];
}
@end

@implementation MainView
- (id)initWithFrame:(CGRect)rect {

    self = [ super initWithFrame: rect ];
    if (nil != self) {
        NSString *audioDeviceName;

        avs = [ AVSystemController sharedAVSystemController ];
        [ avs getActiveCategoryVolume:&_volume andName:
            &audioDeviceName ];

        textView = [ [ UITextView alloc ] initWithFrame: rect ];
        [ self displayVolume ];
        [ self addSubview: textView ];

        [ [ NotificationCenter defaultCenter ] addObserver: self
            selector:@selector(volumeChange:)
            name:
                @"AVSystemController_SystemVolumeDidChangeNotification"
            object: avs ];
    }
}
```

```
return self;
}

- (void)displayVolume
{
    NSString *text;

    text = [ [ NSString alloc ]
              initWithFormat: @"Volume is set to %f", _volume ];
    [ textView setText: text ];
}

- (void)volumeChange:(NSNotification *)notification {
    AVSystemController *avsc = [ notification object ];
    NSString *audioDeviceName;

    [ avsc getActiveCategoryVolume:&_volume andName:&audioDeviceName ];
    [ self displayVolume ];
}

- (void)dealloc
{
    [ self dealloc ];
    [ super dealloc ];
}

@end
```

Как это работает

Пример с уровнем громкости работает следующим образом:

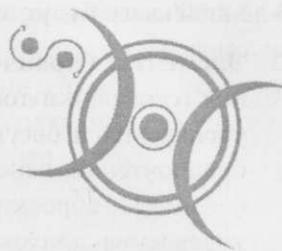
1. При порождении приложения создается объект `MainView` и вызывается его метод `initWithFrame`.
2. Метод `initWithFrame` создает экземпляр `AVSystemController` и делегирует наблюдателю полномочия уведомлять метод `volumeChange` об изменениях системного уровня громкости.

3. Уровень громкости один раз считывается в начале, и отображается текстовый вид.
4. Когда пользователь нажимает одну из кнопок регулировки уровня громкости на телефоне, наблюдатель уведомляет метод `volumeChange`.
5. Метод `volumeChange` считывает новый системный уровень громкости и вызывает метод под названием `displayVolume` для обновления выводимого текста.

Для дальнейшего изучения

Проверьте наличие прототипов `Audio Toolbox` в папке `include` вашего пакета инструментов. Вы обнаружите их в папке `/toolchain/sys/usr/include/AudioToolbox`.

ГЛАВА 7



Проектирование в UIKit для опытных пользователей

В *главе 3* вы познакомились с платформой UIKit, являющейся сердцем всех приложений GUI на iPhone. Данная глава описывает наиболее эстетически богатые компоненты платформы UIKit на iPhone и показывает вам, как сделать ваше собственное программное обеспечение таким же эффектным, как и поставляемые Apple приложения.

Помните, что для использования платформы UIKit ваше приложение должно подключиться к ней. Как и любая другая платформа, UIKit является разделяемым объектом. При использовании пакета инструментов UIKit может быть присоединена путем добавления следующих аргументов к аргументам вашей командной строки:

```
$ arm-apple-darwin9-gcc -o MyApp MyApp.m -lobjc \
    -framework CoreFoundation \
    -framework Foundation \
    -framework UIKit
```

Если вы используете make-файл, как было показано в *главе 2*, то добавьте платформу UIKit в раздел флагов компоновщика:

```
LDFLAGS = lobjc \
    -framework CoreFoundation \
    -framework Foundation \
    -framework UIKit
```

В данной главе будут освещены следующие расширенные компоненты UIKit.

- ❑ **Элементы управления.** UIKit предоставляет набор элементов управления (controls), в том числе переключатели, сегментированные элементы управления и бегунки. Элементы управления используются в таблицах предпочтений, панелях навигации и других визуальных элементах. Класс `UIControl` спроектирован как многофункциональный класс элементов управления, допускающий подключение к различным типам объектов.
- ❑ **Таблицы предпочтений.** Специально для управления настройками программ был создан особый класс. Таблицы предпочтений (preferences tables) предоставляют методы для связывания элементов управления и позволяют логически группировать схожие параметры. Вид таблицы предпочтений — это класс типа таблицы, связывающий воедино множество различных типов отдельных ячеек и элементов управления в хорошо продуманное окно.
- ❑ **Индикаторы прогресса.** Индикаторы прогресса (progress indicators) сообщают пользователю о том, что какая-либо операция находится в процессе выполнения, и передают ее статус в виде вращающихся значков и термометров. Приложение может сообщать индикатору, когда начинать и когда останавливаться, а также может управлять завершением индикатора выполнения.
- ❑ **Обработка изображений.** UIKit предоставляет множество классов для обработки и отображения изображений. Эти классы могут загружать наиболее распространенные типы изображений, а также отображать, изменять, наслаивать и отсекают их в любом месте экрана. Классы изображений предназначены для статических изображений, которым не требуется большой объем анимации. О работе с высококачественной графикой в играх читайте в *главе 5*.
- ❑ **Списки разделов.** При работе с большими сгруппированными списками данных, возможно, вы посчитаете необходимым упорядочить информацию по категориям. Списки разделов (section lists) используются в приложениях мобильных телефонов для сортировки контактов и песен, а также могут быть использованы для группировки элементов любого типа в категории с задаваемыми заголовками. Кроме того, для быстрого перехода к первой букве заголовка раздела можно использовать алфавитную строку прокрутки типа Rolodex.
- ❑ **Выборщики.** Формально выборщики (Pickers) не являются элементами управления, они предоставляют унифицированный метод ввода парамет-

ров, выбранных из списка. Выборщики представляют списки в виде вращающихся циферблатов, которые могут быть специально настроены на различное поведение. Выборщики даты и времени являются наиболее специализированными версиями этого класса, позволяющими осуществлять выбор собственных дат, времени и временных периодов.

- **Панели инструментов.** Панели инструментов (ранее называемые кнопочными панелями (button bars)) — это панели значков или текстовые панели, отображаемые внизу приложения. Панели инструментов (toolbars) являются предпочтительным методом логического разделения схожих видов различных данных или предоставления разных режимов функциональности в рамках приложения. Приложение iPod использует панели инструментов для отделения друг от друга листов воспроизведения, исполнителей, песен и видео, а приложение телефона использует панель инструментов для разделения различных функций телефона (клавиатуры, контактов и т. д.).
- **Изменения ориентации.** Класс `UIHardware` из UIKit предоставляет доступ к датчику положения (orientation sensor). Это позволяет разработчику узнавать, когда iPhone переключается между альбомной и книжной ориентациями, и под каким углом держат телефонную трубку. Вы можете напрямую считывать показания акселерометра iPhone для определения небольших изменений угла, или же использовать более простой API для получения информации об общей ориентации телефонной трубки.
- **Виды Web-документа и прокрутки.** Класс вида Web-документа (Web document view) является встроенным в платформу UIKit и позволяет приложениям отображать Web-страницы или локальные файлы в рамках окна. Это очень мощное средство для сетевых инструментов, которые могут выбрать использование Web-страниц для обновления окна "последних новостей" или отображения другой информации. Web-виды могут также отображать небольшие PDF-файлы и другие файлы, размещенные как локально, так и удаленно.

Виды Web-документа, как и многие другие объекты с большим количеством содержимого, сильно зависят от прокруток (scrollers). Прокрутки предоставляют потенциально огромное поле для размещения вида и позволяют частям этого вида быть прокрученными в пределах небольшой области окна. Вы косвенно использовали их в таблицах и списках, а здесь увидите, как они работают.

Элементы управления

Элементы управления (controls) — это разнообразные служебные классы, созданные для расширения возможностей пользовательского интерфейса приложений. UIKit поддерживает множество различных типов элементов управления. Некоторые из них являются весьма характерными для определенных приложений (например, `UIScrubberControl`), в то время как другие достаточно глубоко скрыты внутри классов более высокого уровня, которые не требуется использовать напрямую большинству разработчиков (например, `UIRemoveControl`).

Элементы управления, используемые в iPhone, существенно отличаются от используемых в настольных приложениях. Элементы управления настольных систем, например, флажки и переключатели, просто не будут помещаться на устройстве с высоким разрешением и ограниченной точностью сенсорного экрана (т. е. без пера). Для каждого используемого элемента управления настольных систем был создан отдельный элемент управления для iPhone.

Элементы управления — это практическое расширение классов, порожденных от `UITableViewCell`, особенно ячейки таблицы, ячейки таблицы предпочтений и другие подобные классы. Некоторые также могут быть использованы с панелями навигации и объектами других типов. Существуют три компактных элемента управления общего назначения для большинства приложений. Они будут описаны в этом разделе. Другие объекты наподобие элементов управления, например, индикаторы прогресса и выборщики, отдельно рассмотрены в соответствующих разделах этой главы.

Элементы управления порождены от базового класса `UIControl`, который в свою очередь порожден от класса `UIView`. Это означает, что он обладает множеством таких же свойств, как и другие классы вида, с которыми вы уже познакомились. Элементы управления инициализируются с областью вида и ведут себя по большей части так же, как и объекты `UIView`.

Сегментированные элементы управления

Сегментированные элементы управления (segmented controls) заменяют переключатели из настольных операционных систем, а вместо них предоставляют интерфейс, схожий с тем, который имеется на передней панели посудомоечной машины. Пользователь видит панель инструментов, на которой нажатие одной кнопки приводит к поднятию всех остальных. Сегментированные эле-

менты управления полезны при ограниченном количестве связанных вариантов выбора, доступных для одного элемента.

Создание элемента управления

Сегментированный элемент управления инициализируется с областью вида. Координаты фрейма это смещение вида, хранящего данный элемент управления, являющийся обычно таблицей предпочтений или панелью навигации:

```
UISegmentedControl *segCtl = [ [UISegmentedControl alloc]
initWithFrame:CGRectMake(70.0, 8.0, 180.0, 30.0)
withStyle: 0
withItems: NULL ];
```

В зависимости от того, где используются элементы управления, можно выбрать один из трех различных стилей сегментированного элемента управления (табл. 7.1).

Таблица 7.1

Стиль	Описание
0	Большие белые кнопки с серой границей, подходит для ячеек предпочтений
1	Большие белые кнопки с черной границей, подходит для ячеек таблицы
2	Маленькие серые кнопки, подходит для панелей навигации

Каждый сегмент в пределах сегментированного вида представлен кнопкой, содержащей метку или изображение. Для каждого возможного варианта выбора должен быть создан отдельный сегмент. Сегментов может быть столько, сколько поместится на экране, но в один момент времени пользователь может выбрать только один сегмент. Варианты выбора для элемента управления "mood" могут выглядеть так, как в приведенном далее фрагменте кода:

```
[ segCtl insertSegment:0 withTitle:@"Happy" animated: YES ];
[ segCtl insertSegment:1 withTitle:@"Sad" animated: YES ];
[ segCtl insertSegment:2 withTitle:@"Mad" animated: YES ];
```

Данный код добавляет на сегментированный элемент управления три кнопки: **Happy**, **Sad** и **Mad**. Каждой кнопке присваивается ее собственный номер сегмента: 0, 1, 2.

Сегменты могут быть удалены.

Чтобы удалить какой-либо отдельный сегмент, воспользуйтесь методом `removeSegmentAtIndex:`

```
[ segCtl removeSegmentAtIndex: 0 animated: YES ];
```

Чтобы удалить все сегменты за раз, вызовите метод `removeAllSegments`. Это приведет к тому, что элемент управления скроет свои кнопки:

```
[ segCtl removeAllSegments ];
```

Если когда-либо понадобится изменить заголовок какой-либо кнопки, воспользуйтесь методом `setTitle:`

```
[ segCtl setTitle:@"Glad" forSegmentAtIndex: 0 ];
```

Изображения

Помимо текста сегментированные элементы управления могут также содержать на кнопках изображения. Любые используемые изображения должны храниться в программной папке приложения, как это обсуждалось в *главе 2*. Изображение может быть добавлено к существующему сегменту с помощью метода `setImage:`

```
[ segCtl setImage: [ UIImage imageNamed:@"happy.png" ]
  forSegmentAtIndex: 0
];
```

Или же, если сегмент еще не был добавлен, другая версия метода `insertSegment` позволит указать изображение после добавления сегмента:

```
[ segCtl insertSegment: 0
  withImage: [ UIImage imageNamed:@"happy.png" ]
  animated: YES
];
```

Сам по себе элемент управления ничего не делает относительно того, что касается масштабирования изображения, поэтому он попытается отобразить ваше изображение на кнопке; даже в том случае, если это изображение слишком велико. Поэтому нужно аккуратно разрабатывать изображения, чтобы они помещались на кнопке.

Мгновенные щелчки

По умолчанию сегментированный элемент управления позволяет пользователю выбирать один вариант за раз и удерживать эту кнопку нажатой до тех пор, пока не будет выбран другой вариант. Такое поведение можно немного поменять на автоматическое отпусканье кнопки сразу же после ее нажатия. Для реализации данного поведения воспользуйтесь методом `setMomentary:`:

```
[ segCtl setMomentary: YES ];
```

Отображение элемента управления

После того как элемент управления будет настроен, его можно отобразить путем добавления в подвид к объекту любого типа, который сможет его принять. Такими объектами могут стать ячейки таблицы, панели навигации и другие объекты вида:

```
[ parentView addSubview: segCtl ];
```

Считывание элемента управления

Чтобы прочесть текущее значение сегментированного элемента управления, воспользуйтесь методом `selectedSegment`. Этот метод возвращает целое число, соответствующее номеру сегмента, выделенному на текущий момент. Данное значение основано на числе, присвоенном ему, когда он впервые был вставлен в элемент управления:

```
int x = [ segCtl selectedSegmentIndex ];
```

Простого считывания значения элемента управления будет вполне достаточно для большинства случаев, например, как при работе с таблицами предпочтений, которым необходимо считывать его, только когда пользователь уходит со страницы. В некоторых же случаях при нажатии кнопки необходимо отправлять уведомление.

Для такой дополнительной функциональности создайте подкласс `UISegmentedControl` и подмените методы `mouseDown` или `mouseUp` вашего элемента управления (в зависимости от того, когда вы хотите получать уведомление). Эти события можно использовать для инициирования чтения или любого другого необходимого действия в момент нажатия кнопки. Вот шаблон для этого:

```
@interface MySegmentedControl : UISegmentedControl
```

```

- (void)mouseDown:(struct __GSEvent *)event;
- (void)mouseUp:(struct __GSEvent *)event;
@end

```

Теперь подмените метод `mouseDown` или метод `mouseUp` так, чтобы значение можно было читать тогда, когда кнопка в этом классе нажата или отпущена:

```

@implementation MySegmentedControl
- (void)mouseDown:(struct __GSEvent *)event {
    int x = [ self selectedSegment ];
    /* Здесь что-нибудь сделайте */
}
@end

```

Переключающий элемент управления

Так же, как сегментированный элемент управления заменяет переключатель, переключающий элемент управления (switch control) заменяет флажок. Переключающие элементы управления используются в панелях предпочтений для включения или выключения каких-либо возможностей.

Переключающий элемент управления значительно проще в использовании, но, тем не менее, может быть настроен до определенной степени.

Создание элемента управления

Переключающий элемент управления инициализируется с помощью стандартного метода `initWithFrame`. Данный метод определяет его размер и координаты относительно принимающего класса, например, ячейки таблицы или панели навигации:

```

UISwitchControl *switchControl = [ [ UISwitchControl alloc ]
initWithFrame: CGRectMake(170.0f, 5.0f, 120.0f, 30.0f)
];

```

Если такой переключатель является достаточно опасным и может повлиять на быстродействие системы или же послужить причиной других серьезных последствий, то лучше пометить его предупреждающими цветами. Можно заставить объект `UISwitchControl` использовать альтернативный набор цветов, отображаться при активации в ярко-оранжевом цвете.

Для этого воспользуйтесь при создании элемента управления методом `setAlternateColors:`

```
[ switchControl setAlternateColors: YES^];
```

Отображение элемента управления

После создания и инициализации элемента управления его можно отобразить, добавив его к объекту вида, например, ячейке таблицы или панели навигации:

```
[ tableViewCell addSubview: switchCtl ];
```

Считывание элемента управления

Переключающий элемент управления возвращает значение с плавающей запятой 0,0, если он выключен, и ненулевое значение, если он включен. Это значение можно получить путем вызова метода `value` данного объекта:

```
float switchValue = [ switchControl value ];
```

Большинству приложений необходимо считывать переключающее значение при уходе со страницы, например, в таблицах предпочтений. Чтобы считать значение такого элемента управления в момент его переключения, нужно перехватить его событие `mouseDown` или `mouseUp`. Чтобы подменить эти методы, унаследуйте `UISwitchControl`. Например:

```
@interface MySwitchControl : UISwitchControl
{
}
- (void)mouseUp:(struct __GSEvent *)event;
@end
```

Когда такой элемент управления переключается, уведомляется метод `mouseUp`, позволяя тем самым мгновенно предпринять какое-либо действие:

```
@implementation UIMySwitchControl
- (void)mouseUp:(struct __GSEvent *)event {
    [ super mouseUp: event];
    float switchValue = [ self value ];
    /* Здесь что-нибудь сделайте */
}
@end
```

Слайдеры

Слайдеры (slider controls) предоставляют область, в которой пользователь может выбирать с помощью визуальной полосы прокрутки, и могут настраиваться для удовлетворения широкого спектра потребностей. Вы можете задать диапазоны значений бегунка, добавить на концы изображения, а также сделать различные другие настройки внешнего вида. Бегунок идеально подходит для представления вариантов выбора с широким диапазоном численных значений, например, настройка уровня громкости, элементы управления чувствительностью и даже элементы управления, требующие тонкой регулировки. Apple просто обязана достаточно хорошо определить слайдеры, чтобы перенести их в iPhone, поскольку они широко распространены в настольных системах.

Создание элемента управления

Слайдер является стандартным объектом `UIControl` и создается точно так же, как и сегментированный или переключающий элементы управления. На самом деле переключающий элемент управления, рассмотренный в предыдущем разделе, порожден от слайдера несмотря на то, что последний является более сложным:

```
UISliderControl *sliderControl = [ [ UISliderControl alloc ]
initWithFrame:CGRectMake(170.0f, 5.0f, 120.0f, 30.0f)];
```

Диапазон значений такого элемента управления должен задаваться при создании, поэтому вы знаете, какие данные ожидать при возврате. Если вы не укажете диапазона по умолчанию, то будут использованы значения между 0,0 и 1,0:

```
[ sliderControl setMinValue: 0.0 ];
[ sliderControl setMaxValue: 100.0 ];
```

В этот же момент можно задать и значения для полосы прокрутки:

```
[ sliderControl setValue: 50.0 ];
```

Слайдер может отображать изображения на любом своем конце. Это можно задать таким же способом, как и в сегментированном элементе управления. Изображения должны быть скопированы в программную папку приложения, как было рассказано в *главе 2*. Использование изображений приводит к увеличению длины полосы прокрутки, поэтому убедитесь в том, что при вызове

`initWithFrame` вы увеличили размер элемента управления в соответствии с размерами изображений:

```
[ sliderControl setMinValueImage:
    [ UIImage imageNamed:@"min.png" ]
];

[ sliderControl setMaxValueImage:
    [ UIImage imageNamed:@"max.png" ]
];
```

Для элементов управления точностью может оказаться существенным отображать пользователю текущее числовое значение. Для отображения такого значения рядом с полосой прокрутки может быть вызван метод `setShowValue:`

```
[ sliderControl setShowValue: YES ];
```

Отображение элемента управления

Как и все стандартные объекты `UIView` данный элемент управления может быть отображен путем добавления его в качестве подвида к ячейке таблицы, панели навигации или другому подходящему объекту:

```
[ tableViewCell addSubview: sliderControl ];
```

Считывание элемента управления

Слайдер считывает значение с плавающей запятой в пределах диапазона, указанного вами в момент создания данного элемента управления. Само значение можно запросить с помощью метода `value:`

```
float sliderValue = [ sliderControl value ];
```

Большинство приложений считывает значения слайдера только тогда, когда пользователь уходит со страницы, например, как в случае с таблицами предпочтений.

Чтобы считать значение такого элемента управления в момент его изменения, нужно перехватить событие `mouseDown` или событие `mouseUp`. Чтобы подменить эти методы, унаследуйте `UISliderControl`, используя следующий шаблон:

```
#interface MySliderControl : UISliderControl
```

```
{
}
- (void)mouseUp:(struct __GSEvent *)event;
@end
[ super mouseUp: event ];
@implementation MySliderControl
- (void)mouseUp:(struct __GSEvent *)event {
    float x = [ self value ];

    /* Здесь что-нибудь сделайте */
}
@end
```

Для дальнейшего изучения

- В следующем разделе прочитайте, как используются элементы управления в таблицах предпочтений.
- Попробуйте придумать различные типы классов вида, где может пригодиться прикрепление элементов управления.
- Проверьте наличие `UISegmentedControl.h`, `UISwitchControl.h` и `UISliderControl.h` в папке `include` вашего пакета инструментов. Вы обнаружите их в папке `/toolchain/sys/usr/include/UIKit`.

Таблицы предпочтений

Таблицы предпочтений (preferences tables) предоставляют эстетически богатый интерфейс для отображения изменения программных настроек. Эти таблицы можно увидеть в iPhone-приложении **Settings**, но большинство приложений сторонних фирм предоставляют собственный интерфейс настроек, чтобы не допустить внесения изменений во встроенное окружение. Таблицы предпочтений предоставляют ячейки меняющегося размера, способные вмещать элементы управления, текстовые поля и информационный текст. Они также предоставляют механизм для логического объединения схожих предпочтений.

Создание таблицы предпочтений

При реализации класса `UIPreferencesTable` необходимо быть достаточно предусмотрительным, поскольку для запроса информации, необходимой для заполнения таблицы, используется ориентированная на обратный вызов *привязка данных* (data binding). Это делается аналогично объектам `UITableView`, изученным вами в *главе 3*, но на более высоком уровне сложности. Исполняющий класс вызывает в источнике данных целый ряд методов для возвращения информации о таблице предпочтений: количество и размер каждой ячейки, объекты внутри ячеек, а также информацию о логических группировках.

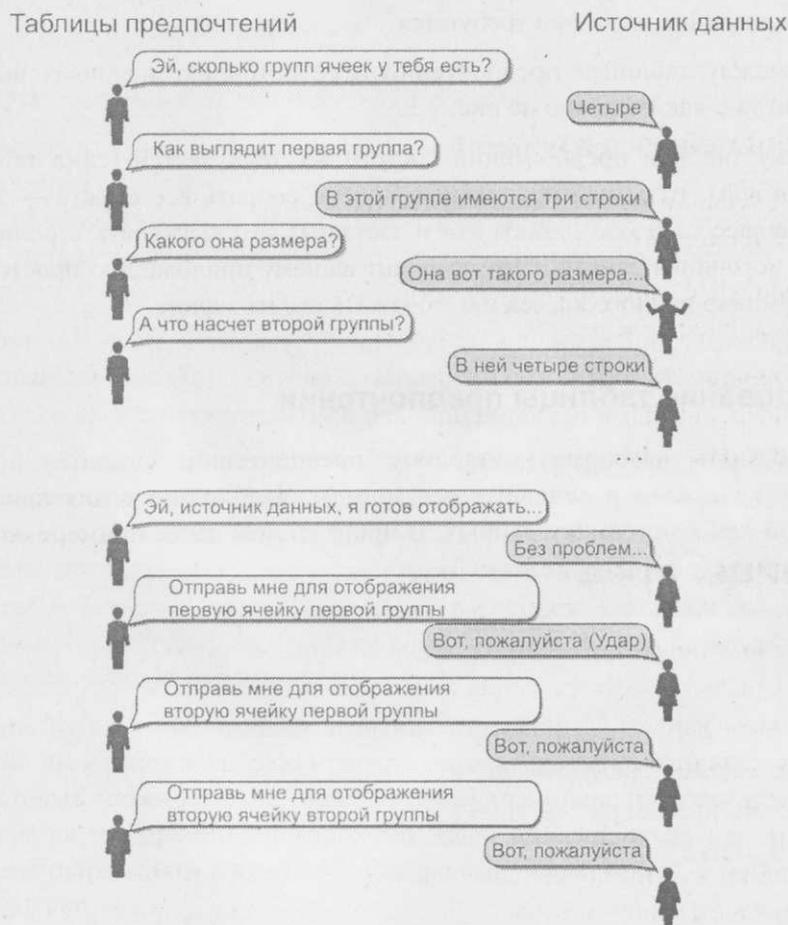


Рис. 7.1. Аналогия диалога между таблицей предпочтений и ее источником данных

К большому разочарованию сообщества открытого кода iPhone все это существенно отличается от объектно-ориентированной модели, в которой каждая ячейка имеет собственные свойства и просто добавляется в качестве объекта. Вместо этого создание всей таблицы предпочтений — достаточно трудоемкое и сложное занятие в противовес традиционно элегантному стилю проектирования.

В качестве резюме стоит сказать, что таблица предпочтений ссылается на страницу настроек. Таблица может иметь множество логических группировок схожих настроек. В рамках каждой группы для отображения пользователю каждой отдельной настройки используется единственная ячейка таблицы. Содержимое ячейки включает в себя необязательные заголовок, текст и элементы управления, если они требуются.

Диалог между таблицей предпочтений и ее источником данных выглядит примерно так, как показано на рис. 7.1.

Поскольку таблица предпочтений состоит из двух частей (сама таблица и данные в ней), то самый правильный способ собрать все вместе — это создать подкласс `UIPreferencesTable` и заставить его выступать в роли собственного источника данных. Это позволит вашему приложению просто создавать экземпляр данного класса и отображать его на экране.

Наследование таблицы предпочтений

Чтобы создать автономную таблицу предпочтений, создайте подкласс `UIPreferencesTable` и включите все методы, необходимые для привязки к ней самой как к источнику данных. В приведенном далее примере создается подкласс `MyPreferencesTable`:

```
@interface MyPreferencesTable : UIPreferencesTable
{
}

/* Методы таблицы предпочтений */
- (id)initWithFrame:(CGRect)rect;
- (void)dealloc;

/* Методы источника данных */
- (int)numberOfGroupsInPreferencesTable:(UIPreferencesTable *)aTable;
```

```
- (UIPreferencesTableCell *)preferencesTable:  
    (UIPreferencesTable *)aTable cellForGroup:(int)group;  
- (float)preferencesTable:(UIPreferencesTable *)aTable  
    heightForRow:(int)row  
    inGroup:(int)group  
    withProposedHeight:(float)proposed;  
- (BOOL)preferencesTable:(UIPreferencesTable *)aTable  
    isLabelGroup:(int)group;  
- (UIPreferencesTableCell *)preferencesTable:  
    (UIPreferencesTable *)aTable  
    cellForRow:(int)row  
    inGroup:(int)group;
```

Для работы с источником данных используются следующие методы:

- `numberOfGroupsInPreferencesTable` — возвращает количество логических групп в таблице. Эти группы включают также группы текстовых меток, используемые для добавления к ячейке небольшого выпуклого текста. Метки групп не должны включаться в групповой подсчет;
- `cellForGroup` — возвращает объект `UIPreferencesTableCell`, чтобы он выступил в роли заголовка группы для заданной группы. Эти ячейки создаются как объекты `UIPreferencesTableCell`, и с помощью метода `setTitle` им присваивается заголовок группы;
- `heightForRow` — возвращает высоту заданной строки. Данный метод снабжается предварительно заданной высотой, которая может быть возвращена как высота по умолчанию. Разумеется, высота любой ячейки определяется разработчиком, вы можете вернуть как предзаданную высоту, так и заданную вами, в зависимости от типа создаваемого объекта. Данный метод также вызывается для группы заголовков, используя при этом в качестве номер строки `-1`;
- `isLabelGroup` — возвращает значение типа `Boolean`, указывающее на то, нужно ли ячейки в заданной группе рассматривать как текстовые метки. Текстовые метки меньше, чем метки групп, и используются для предоставления информационного текста. Если возвращается `YES`, то группа будет отображаться в виде небольшого выпуклого текста, а не в виде белой ячейки;
- `cellForRow` — возвращает объект ячейки предпочтения для любой указанной строки или группы. Каждый раз, когда предпочтение попадает в

область видимости, вызывается данный метод для получения объекта `UIPreferencesTableCell`, представляющего строку. Этот метод вызывается много раз для одной и той же ячейки, поэтому объекты ячейки должны кэшироваться в памяти.

Кэширование ячеек предпочтений

Каждая ячейка в таблице предпочтений "забывается" таблицей после того, как она прокручивается с экрана. Когда она вновь прокручивается и попадает на экран, для данной строки снова вызывается метод `cellForRow`. Это может послужить источником определенных проблем, если вы создаете новую строку каждый раз при вызове метода, а также может существенно замедлить прокрутку.

Иначе говоря, метод `cellForRow` должен быть написан так, чтобы повторяющиеся запросы на одну и ту же ячейку предпочтений возвращали бы указатель на существующую ячейку, а не пересоздавали бы ее каждый раз. Создав матрицу, вы можете хранить указатели на каждую ячейку и возвращать их позднее, чтобы удовлетворить последующим запросам. Создайте матрицу указателей ячеек в вашем подклассе таблицы предпочтений. Она должна будет содержать все указатели на ячейки, которые вы создадите позднее:

```
#define NUM_GROUPS 3
#define CELLS_PER_GROUP 4

UIPreferencesTableCell *cells[NUM_GROUPS][CELLS_PER_GROUP];
UIPreferencesTableCell *groupcell[NUM_GROUPS];
```

Здесь `NUM_GROUPS` и `CELLS_PER_GROUP` представляют количество логических групп в таблице предпочтений и максимальное количество ячеек в любой заданной группе. Эти значения будут использоваться для инициализации матрицы при вызове метода таблицы предпочтений `initWithFrame`, поэтому они задаются здесь как константы.

Инициализация

После создания объекта таблицы предпочтений вы отображаете ее либо путем прикрепления к существующему виду, либо путем перехода к нему из управляющего вида. Это можно сделать, когда пользователь нажимает кноп-

ку настроек, или же автоматически при первом запуске программы. Приведенный далее код порождает созданный вами подкласс `MyPreferencesTable`:

```
MyPreferencesTable *preferencesTable = [ [ MyPreferencesTable alloc ]  
initWithFrame: viewRect ];
```

После создания объекта должен быть вызван его метод `reloadData` для загрузки всех элементов в таблицу предпочтений. Это приведет к тому, что данная таблица вызовет свой источник данных и начнет загрузку информации о группировке и конфигурации ячеек:

```
[ preferencesTable reloadData ];
```

Ячейки таблицы предпочтений

Каждая ячейка в таблице предпочтений создается как объект `UIPreferencesTableCell` или же его подкласс. Ячейки возвращаются посредством метода обратного вызова `cellForRow`, который вы должны написать и который автоматически вызывается классом таблицы предпочтений, как только новые строки отрисовываются на экране. Например, если метод `cellForRow` вызывается, задавая строку 0 группы 0, то данный метод может возвращать ячейку, используя, например, такой код:

```
UIPreferencesTableCell *cell = [ [ UIPreferencesTableCell alloc ] init ];  
[ cell setTitle:@"Music Volume" ];  
[ cell setShowSelection: NO ];  
return cell;
```

Текстовые ячейки

Текстовая ячейка (`text cell`) является подклассом `UIPreferencesTableCell`, но спроектирована для отображения (и, необязательно, — для редактирования) текста. Приведенный далее фрагмент кода — пример того, как создать такую ячейку:

```
UIPreferencesTextTableCell *cell = [ [ UIPreferencesTextTableCell alloc ]  
initWithFrame: viewRect ];  
[ cell setTitle:@"Version" ];  
[ cell setEnabled: NO ];  
[ cell setValue:@"1.0.0" ];  
return cell;
```

Чтобы разрешить редактирование, воспользуйтесь методом `setEnabled`:

```
[ cell setEnabled: YES ];
```

Само значение может быть считано с помощью метода значения:

```
NSString *text = [ cell value ];
```

Элементы управления

Объект `UIPreferencesControlTableCell` порожден от базового класса `UIPreferencesTableCell` и может вмещать элемент управления. Элементы управления могут быть добавлены к обычным объектам `UIPreferencesTableCell` как подвиды, но класс `UIPreferencesControlTableCell` может полностью инкапсулировать элемент управления:

```
UIPreferencesControlTableCell *cell =  
    [ [ UIPreferencesControlTableCell alloc ] init ];  
UISliderControl *musicVolumeControl = [ [ UISliderControl alloc ]  
    initWithFrame:CGRectMake(170.0f, 5.0f, 120.0f, 55.0f) ];  
[ musicVolumeControl setMinValue: 0.0 ];  
[ musicVolumeControl setMaxValue: 10.0 ];  
[ musicVolumeControl setValue: 3.5 ];  
  
[ cell setControl: musicVolumeControl ];
```

Затем на этот элемент управления можно ссылаться из объекта, что освобождает вас от необходимости хранить указатель на этот элемент управления в вашем классе:

```
UISliderControl *musicVolumeControl = [ cell control ];
```

В данном примере создается элемент управления "бегунок" с фреймом в правой части ячейки предпочтений. Затем бегунок определяется как элемент управления данной ячейки. Все это происходит в методе `cellForRow` до возвращения только что созданной ячейки.

Отображение таблицы предпочтений

Таблица предпочтений может быть отображена путем добавления ее как подвида к существующему виду или перехода к ней с `UITransitionView`.

Добавление таблицы предпочтений как подвида происходит следующим образом:

```
[ self addSubview: preferencesTable ];
```

а переход может выглядеть примерно так:

```
[ transitionView transition: 1 toView: preferencesTable ];
```

Пример: настройки игры-стрелялки

Написана космическая игра-стрелялка, и ей необходим набор параметров для управления всем: от уровня громкости звука до отладочных сообщений. В данном примере класс `UIPreferencesTable` наследуется для создания нашего собственного объекта `MyPreferencesTable`. Этот объект служит в качестве таблицы предпочтений и ее собственного источника данных. Он создает каждую ячейку и назначает некоторые из элементов управления, о которых вы узнали в предыдущем разделе. Объект `MyPreferencesTable` построен как независимая таблица предпочтений, которая может быть использована классом `MainView`.

Данный пример может быть построен из командной строки с использованием пакета инструментов следующим образом:

```
$ arm-apple-darwin9-gcc -o MyExample MyExample.m MyPreferencesTable.m \
  -lobjc -framework CoreFoundation -framework UIKit
```

Листинги 7.1 и 7.2 содержат код для приложения и основного вида, а листинги 7.3 и 7.4 создают саму таблицу предпочтений.

Листинг 7.1. Пример таблицы предпочтений (MyExample.h)

```
#import <CoreFoundation/CoreFoundation.h>
#import <UIKit/UIKit.h>
#import "MyPreferencesTable.h"

@interface MainView : UIView
{
    MyPreferencesTable *preferencesTable;
}
- (id)initWithFrame:(CGRect)rect;
- (void)dealloc;
@end
```

```
@interface MyApp : UIApplication
{
    UIWindow *window;
    MainView *mainView;
}
- (void)applicationDidFinishLaunching:(NSNotification *)aNotification;
@end
```

Листинг 7.2. Пример таблицы предпочтений (MyExample.m)

```
#import "MyExample.h"

int main(int argc, char **argv)
{
    return UIApplicationMain(argc, argv, @"MyApp", @"MyApp");
}

@implementation MyApp
- (void)applicationDidFinishLaunching:(NSNotification *)aNotification {
    window = [ [ UIWindow alloc ] initWithContentRect:
        [ UIHardware fullScreenApplicationContentRect ]
    ];

    CGRect rect = [ UIHardware fullScreenApplicationContentRect ];
    rect.origin.x = rect.origin.y = 0.0f;

    mainView = [ [ MainView alloc ] initWithFrame: rect ];

    [ window setContentView: mainView ];
    [ window orderFront: self ];
    [ window makeKey: self ];
    [ window _setHidden: NO ];
}
@end
```

```
@implementation MainView
- (id)initWithFrame:(CGRect)rect {
    self = [super initWithFrame: rect];
    if (nil != self) {
        preferencesTable = [ [ MyPreferencesTable alloc ]
                               initWithFrame: rect ];
        [ preferencesTable reloadData ];
        [ self addSubview: preferencesTable ];
    }

    return self;
}

- (void)dealloc
{
    [ self dealloc ];
    [ super dealloc ];
}
@end
```

Листинг 7.3. Пример таблицы предпочтений (MyPreferencesTable.h)

```
#import <UIKit/UIKit.h>
#import <UIKit/UIPreferencesTable.h>
#import <UIKit/UIPreferencesTextTableCell.h>
#import <UIKit/UISwitchControl.h>
#import <UIKit/UISegmentedControl.h>
#import <UIKit/UISliderControl.h>

#define NUM_GROUPS 3
#define CELLS_PER_GROUP 4

@interface MyPreferencesTable : UIPreferencesTable
{
    UIPreferencesTableCell *cells[NUM_GROUPS][CELLS_PER_GROUP];
    UIPreferencesTableCell *groupcell[NUM_GROUPS];
}
```

```

UISliderControl *musicVolumeControl;
UISliderControl *gameVolumeControl;
UISegmentedControl *difficultyControl;

UISliderControl *shipStabilityControl;
UISwitchControl *badGuyControl;
UISwitchControl *debugControl;
}

- (id)initWithFrame:(CGRect)rect;
- (int)numberOfGroupsInPreferencesTable:(UIPreferencesTable *)aTable;
- (UIPreferencesTableCell *)preferencesTable:
    (UIPreferencesTable *)aTable
    cellForGroup:(int)group;
- (float)preferencesTable:(UIPreferencesTable *)aTable
    heightForRow:(int)row
    inGroup:(int)group
    withProposedHeight:(float)proposed;
- (BOOL)preferencesTable:(UIPreferencesTable *)aTable
    isLabelGroup:(int)group;
- (UIPreferencesTableCell *)preferencesTable:
    (UIPreferencesTable *)aTable
    cellForRow:(int)row
    inGroup:(int)group;
@end

```

Листинг 7.4. Пример таблицы предпочтений (MyPreferencesTable.h)

```

#import "MyPreferencesTable.h"

@implementation MyPreferencesTable
- (id)initWithFrame:(CGRect)rect {
    self = [super initWithFrame:rect];
    if (nil != self) {
        int i, j;

```

```
for(i=0;i<NUM_GROUPS;i++) {
    groupcell[i] = NULL;
    for(j=0;j<CELLS_PER_GROUP;j++)
        cells[i][j] = NULL;
}

[ self setDataSource: self ];
[ self setDelegate: self ];
}

return self;
}

- (int)numberOfGroupsInPreferencesTable:(UIPreferencesTable *)aTable {

    /* Количество логических групп, включая метки */
    return NUM_GROUPS;
}

- (int)preferencesTable:(UIPreferencesTable *)aTable
  numberOfRowsInGroup:(int)group
{
    switch (group) {
        case(0):
            return 4;
            break;
        case(1):
            return 4;
            break;
        case(2):
            return 1; /* Группа текстовых меток */
            break;
    }
}
}
```

```
- (UIPreferencesTableCell *)preferencesTable:
(UIPreferencesTable *)aTable
cellForGroup:(int)group
{
    if (groupcell[group] != NULL)
        return groupcell[group];

    groupcell[group] = [ [ UIPreferencesTableCell alloc ] init ];
    switch (group) {
        case (0):
            [ groupcell[group] setTitle: @"General Settings" ];
            break;
        case (1):
            [ groupcell[group] setTitle: @"Advanced Settings" ];
            break;
    }
    return groupcell[group];
}

- (float)preferencesTable:(UIPreferencesTable *)aTable
heightForRow:(int)row
inGroup:(int)group
withProposedHeight:(float)proposed
{
    /* Возвращаем высоту заголовков группы */
    if (row == -1) {
        if (group < 2)
            return 40;
    }

    /* Сегментированные элементы управления больше, чем остальные */
    if (group == 0 && row == 2)
        return 65.0;

    return proposed;
}
```

```
- (BOOL)preferencesTable:(UIPreferencesTable *)aTable
isLabelGroup:(int)group
{
    if (group == 2)
        return YES;
    return NO;
}

- (UIPreferencesTableCell *)preferencesTable:
    (UIPreferencesTable *)aTable
cellForRow:(int)row
inGroup:(int)group
{
    UIPreferencesTableCell *cell;

    if (cells[group][row] != NULL)
        return cells[group][row];

    cell = [ [ UIPreferencesTableCell alloc ] init ];
    [ cell setEnabled: YES ];

    switch (group) {
        case (0):
            switch (row) {
                case (0):
                    [ cell setTitle:@"Music Volume" ];
                    musicVolumeControl = [ [ UISliderControl alloc ]
                        initWithFrame: CGRectMake(170, 5, 120, 55)
                    ];
                    [ musicVolumeControl setMinValue: 0.0 ];
                    [ musicVolumeControl setMaxValue: 10.0 ];
                    [ musicVolumeControl setValue: 3.5 ];
                    [ cell addSubview: musicVolumeControl ];
                    break;
                case (1):
                    [ cell setTitle:@"Game Sounds" ];
```

```
gameVolumeControl = [ [ UISliderControl alloc ]
    initWithFrame: CGRectMake(170, 5, 120, 55)
];
[ gameVolumeControl setMinValue: 0.0 ];
[ gameVolumeControl setMaxValue: 10.0 ];
[ gameVolumeControl setValue: 5.0 ];
[ cell addSubview: gameVolumeControl ];
break;
case (2):
    [ cell setTitle:@"Difficulty" ];
    difficultyControl = [ [ UISegmentedControl alloc ]
        initWithFrame: CGRectMake(170, 5, 120, 55) ];
    [ difficultyControl insertSegment:0
        withTitle:@"Easy" animated: NO ];
    [ difficultyControl insertSegment:1
        withTitle:@"Hard" animated: NO ];
    [ difficultyControl selectSegment: 0 ];

    [ cell addSubview: difficultyControl ];
    break;
case (3):
    [ cell release ];
    cell = [ [ UIPreferencesTextTableCell alloc ]
        init ];
    [ cell setTitle: @"Cheat Code" ];
    [ cell setEnabled: YES ];
    [ cell setValue: @"None" ];
    break;
}
break;
case (1):
    switch (row) {
        case (0):
            [ cell setTitle:@"Ship Stability" ];
            shipStabilityControl = [
```

```
        [ UISliderControl alloc ]
        initWithFrame: CGRectMake(170, 5, 120, 55)
    ];
    [ shipStabilityControl setMinValue: 0.0 ];
    [ shipStabilityControl setMaxValue: 100.0 ];
    [ shipStabilityControl setValue: 45.0 ];
    [ shipStabilityControl setShowValue: YES ];
    [ cell addSubview: shipStabilityControl ];
    break;
case (1):
    [ cell setTitle:@"Extra Bad Guys" ];
    badGuyControl = [ [ UISwitchControl alloc ]
        initWithFrame:CGRectMake(170, 5, 120, 30)
    ];
    [ badGuyControl setValue: 0.0 ];
    [ cell addSubview: badGuyControl ];
    break;
case (2):
    [ cell setTitle:@"Debugging" ];
    debugControl = [ [ UISwitchControl alloc ]
        initWithFrame:CGRectMake(170, 5, 120, 30)
    ];
    [ debugControl setValue: 0.0 ];
    [ debugControl setAlternateColors: YES ];
    [ cell addSubview: debugControl ];
    break;
case (3):
    [ cell release ];
    cell = [ [ UIPreferencesTextTableCell alloc ]
        init ];
    [ cell setTitle: @"Version" ];
    [ cell setEnabled: NO ];
    [ cell setValue: @"1.0.0" ];
    break;
}
```

```
        break;
    case (2):
        [ cell setTitle:
            @"Settings will take effect on the next restart"
        ];
        break;
    }

    [ cell setShowSelection: NO ];
    cells[group][row] = cell;
    return cell;
}
@end
```

Как это работает

Вы только что прочитали полное приложение, отображающее таблицу предпочтений. Вот как оно работает:

1. При порождении приложения создается объект `MainView`, который служит в качестве управляющего вида для приложения.
2. Классом приложения вызывается метод `initWithFrame` объекта `MainView`. Тем самым порождается объект `MyPreferencesTable` с именем `preferencesTable` и вызывается его метод `initWithFrame`.
3. Собственный метод `initWithFrame` таблицы предпочтений инициализирует внутренний кэш таблицы и определяет сам себя в качестве собственного источника данных для этой таблицы.
4. Объект `mainView` вызывает метод `reloadData` объекта `preferenceTable`. Поскольку мы не подменили `reloadData`, то вызывается метод родительского класса `UIPreferencesTable`. Это порождает взаимодействие с источником данных путем вызова различных методов источника данных. Таблица предпочтений общается с источником данных, чтобы определить основную конструкцию и геометрию таблицы.
5. Объект `preferenceTable` добавлен как подвид `mainView`, где основная платформа управления окнами вызывает свои процедуры отрисовки.
6. Для всех строк, видимых на экране, вызывается метод `cellForRow` таблицы предпочтений.

7. Сначала метод `cellForRow` проверяет, существует ли уже указатель для адресуемой ячейки предпочтений, и если не существует, то создает новый объект `UIPreferencesTableCell`. Заголовок и параметры задаются на основе количества строк и групп, и все элементы управления создаются и добавляются как подвиды ячейки предпочтений.
8. Если создается новая ячейка, возвращается ее объект. Иначе возвращается кэшированный указатель на существующий объект. В любом случае внутренне вызываются низкоуровневые процедуры прорисовки объекта, и он прорисовывается на экране.

Для дальнейшего изучения

Теперь, когда вы получили представление о том, как работают таблицы предпочтений, попробуйте выполнить некоторые упражнения для более близкого знакомства с ними.

- Измените пример так, чтобы добавить действительные переменные для каждого из элементов управления, имеющих собственные методы считывания и методы задания свойств, так чтобы их значения можно было бы обменивать с объектом `MainView`.
- Воспользуйтесь примерами из главы 3 для построения приложения с двумя видами: один табличный или текстовый вид, а другой — таблица предпочтений. Назначьте какую-либо кнопку панели навигации для перехода пользователей к таблице предпочтений, если они нажимают кнопку **Settings**, и обратно к текстовому виду, если они нажимают кнопку **Back**.
- Проверьте наличие `UIPreferencesTable.h`, `UIPreferencesTextTableCell.h`, `UIPreferencesControlTableCell.h` и `UIPreferencesTableCell.h` в папке `include` вашего пакета инструментов. Вы найдете их в папке `/toolchain/sys/include/UIKit`.

Индикаторы прогресса

Индикаторы прогресса (progress indicators) сообщают пользователю о том, что некоторая операция находится в процессе выполнения. Существуют два типа индикаторов, поддерживаемых UIKit:

- `UIProgressIndicator` — данный класс представляет собой вращающуюся анимацию наподобие часов. Такого типа анимацию вы можете видеть, к

примеру, при включении поддержки WiFi или Bluetooth на iPhone, или же при загрузке вашего настольного компьютера Mac;

- `UIProgressbar` — данный класс представляет собой анимацию наподобие показаний термометра, позволяя, таким образом, приложению сообщать, сколько еще времени осталось до момента завершения операции.

Оба типа индикаторов порождены от базового класса `UIView`. Это означает, что они могут наслаиваться поверх текстовых видов, листов предупреждений, большинства типов табличных ячеек и любых других объектов, порождаемых от `UIView`.

UIProgressIndicators: то, что вертится

Класс `UIProgressIndicators` является простым анимационным классом, достаточно небольшим, чтобы прикрепляться к практически любому объекту `UIView`, включая ячейки таблиц и листы предупреждений. Такой индикатор отображает анимацию наподобие часов: метки делений, совершающие обороты по кругу. Такой индикатор создается с фреймом, определяющим размер индикатора и координаты относительно вида, к которому он прикрепляется:

```
UIProgressIndicator *progressView = [ UIProgressIndicator alloc ]
initWithFrame: CGRectMake(0, 0, 32, 32) ];
```

Данный индикатор поддерживает два стиля: белый (0) и темно-серый (1). Какой из этих стилей является более подходящим, зависит от типа объекта, к которому прикрепляется индикатор. Стиль устанавливается путем вызова метода `setStyle` объекта:

```
[ progressView setStyle: 0 ];
```

Для того чтобы задать количество времени, необходимое для одного полного оборота меток по кругу, используйте метод `setAnimationDuration`. Это значение задается в секундах и должно соответствовать ожидаемой длине операции. Более быстрые операции могут использовать более быстрые обороты по кругу, а те операции, которые могут занимать более минуты, должны использовать более медленные вращения:

```
[ progressView setAnimationDuration: 1.0 ];
```

Объект индикатора прогресса может быть добавлен к любому существующему объекту вида. Например, код добавления его к листу предупреждений может выглядеть так:

```
[ alertSheet addSubview: progressView ];
```

Чтобы запустить или остановить анимацию, воспользуйтесь методами `startAnimation` и `stopAnimation`:

```
[ progressView startAnimation ];  
[ progressView stopAnimation ];
```

Пример: простой вращающийся индикатор

Данный пример иллюстрирует построение и отображение вида `UIProgressIndicator`. Пример создает объект индикатора и прикрепляет его к основному виду. Затем создается объект `NSTimer` для прекращения анимации после пяти секунд. Объект `NSTimer` является частью платформы `Foundation` и может находиться в документации `Cocoa` от Apple на Web-узле `Apple Developer Connection`.

Чтобы скомпилировать этот пример из командной строки, воспользуйтесь пакетом инструментов следующим образом:

```
$ arm-apple-darwin9-gcc -o MyExample MyExample.m -lobjc   
-framework UIKit -framework Foundation
```

Листинги 7.5 и 7.6 содержат соответствующий код.

Листинг 7.5. Пример индикатора прогресса (MyExample.h)

```
#import <UIKit/UIKit.h>  
#import <UIKit/UIProgressIndicator.h>  
  
UIProgressIndicator *progressView;  
  
@interface MainView : UIView  
{  
}  
- (id)initWithFrame:(CGRect) frame;  
- (void)dealloc;  
@end  
  
@interface MyApp : UIApplication  
{  
    UIWindow *window;  
}
```

```
    MainView *mainView;
}
- (void)applicationDidFinishLaunching:(NSNotification *)aNotification;
@end
```

Листинг 7.6. Пример индикатора прогресса (MyExample.m)

```
#import "MyExample.h"

int main(int argc, char **argv)
{
    return UIApplicationMain(argc, argv, @"MyApp", @"MyApp");
}

@implementation MyApp

- (void)applicationDidFinishLaunching:(NSNotification *)aNotification {
    window = [ [ UIWindow alloc ] initWithContentRect:
        [ UIScreen fullScreenApplicationContentRect ]
    ];

    CGRect rect = [ UIScreen fullScreenApplicationContentRect ];
    rect.origin.x = rect.origin.y = 0.0f;

    mainView = [ [ MainView alloc ] initWithFrame: rect ];
    [ window setContentView: mainView ];
    [ window orderFront: self ];
    [ window makeKey: self ];
    [ window _setHidden: NO ];

    NSTimer *timer = [ NSTimer scheduledTimerWithTimeInterval: 5.0
        target: self
        selector: @selector(handleTimer:)
        userInfo: nil
        repeats: NO ];
}
```

```
- (void) handleTimer: (NSTimer *) timer
{
    [ progressView stopAnimation ];
}
@end

@implementation MainView
- (id) initWithFrame: (CGRect) rect {

    self = [ super initWithFrame: rect ];
    if (nil != self) {
        CGRect progressRect = rect;
        progressRect.size.width = 32;
        progressRect.size.height = 32;

        progressView = [ [ UIProgressIndicator alloc ]
            initWithFrame: progressRect ];
        [ self addSubview: progressView ];
        [ progressView setAnimationDuration: 1.0 ];
        [ progressView startAnimation ];
    }
    return self;
}

- (void) dealloc
{
    [ self dealloc ];
    [ super dealloc ];
}
@end
```

Как это работает

Вот как работает пример индикатора прогресса:

1. При порождении приложения создается объект `MainView` и вызывается его метод `initWithFrame`.

2. Метод `initWithFrame` создает объект `UIProgressIndicator` и прикрепляет его к основному виду. Анимация устанавливается на использование оборотов за одну секунду, а затем анимация стартует.
3. Создается объект `NSTimer` с триггером в 5 секунд. Как только это время истекает, получает уведомление метод `handleTimer`, который отключает анимацию, что приводит к исчезновению индикатора.

UIProgressBar: когда вращающиеся индикаторы не подходят

Объект `UIProgressBar` является близким родственником объекту `UIProgressIndicator`. Вместо вращающейся анимации класс строки прогресса рисует индикатор наподобие термометра и предоставляет интерфейс для определения у него крайнего уровня в соответствии со временем выполнения операции вашим приложением. Преимущество использования строки прогресса состоит в том, что он может более-менее точно отражать количество действительно выполненной приложением работы.

Чтобы создать строку прогресса, метод инициализации класса включает фрейм, определяющий размер строки и начало координат области отображения:

```
UIProgressBar progressBar = [ [ UIProgressBar alloc ]
initWithFrame: CGRectMake(0, 0, 320, 32) ];
```

Данный индикатор поддерживает два стиля: белый (0) и темно-серый (1). Стиль может быть задан с помощью метода `setStyle` в соответствии с цветом объекта, к которому прикрепляется индикатор:

```
[ progressBar setStyle: 0 ];
```

Чтобы отобразить строку прогресса, добавьте ее к существующему объекту `UIView`. Например, код прикрепления ее к листу предупреждений может выглядеть следующим образом:

```
[ alertSheet addSubview: progressBar ];
```

После того как строка прогресса отобразится на экране, ее прогресс может быть установлен приложением, чтобы показать, где именно оно находится в процессе выполнения операции. После установки уровня делается вызов метода `updateIfNecessary` строки, чтобы обеспечить продвижение строки

прогресса по экрану. Тип значения прогресса — `double` (число с плавающей запятой двойной точности), а само значение находится в диапазоне между 0,0 и 1,0:

```
[ progressBar setProgress: 0.5 ];  
[ progressBar updateIfNeeded ];
```

Пример: усовершенствованная строка прогресса

В данном примере строка прогресса создается и добавляется к основному виду. Объект `NSTimer` используется для запуска метода обновления каждую десятую долю секунды, что увеличивает строку прогресса на 1% (0,01).

Хотя пример останавливается на уровне 1,0, строка прогресса не делает каких-либо попыток убедиться в том, что переданное ей значение является корректным. Передача значения, большего 1,0, приведет к тому, что строка прогресса будет заполнена сверх своих границ. Конечно, в некоторых случаях это может оказаться вполне полезным эффектом.

Чтобы скомпилировать данный пример из командной строки, воспользуйтесь пакетом инструментов следующим образом:

```
$ arm-apple-darwin9-gcc -o MyExample MyExample.m -lobjc  
-framework UIKit -framework Foundation
```

В листингах 7.7 и 7.8 содержится соответствующий код.

Листинг 7.7. Пример строки прогресса (MyExample.h)

```
#import <UIKit/UIKit.h>  
#import <UIKit/UIProgressbar.h>  
  
UIProgressbar *progressView;  
  
@interface MainView : UIView  
{  
  
}  
- (id)initWithFrame:(CGRect) frame;  
- (void)dealloc;  
@end
```

```
@interface MyApp : UIApplication
{
    UIWindow *window;
    MainView *mainView;
}
- (void)applicationDidFinishLaunching:(NSNotification *)aNotification;
@end
```

Листинг 7.8. Пример строки прогресса (MyExample.m)

```
#import "MyExample.h"

int main(int argc, char **argv)
{
    return UIApplicationMain(argc, argv, @"MyApp", @"MyApp");
}

double progress = 0.0;

@implementation MyApp

- (void)applicationDidFinishLaunching:(NSNotification *)aNotification {
    window = [ [ UIWindow alloc ] initWithContentRect:
        [ UIScreen fullScreenApplicationContentRect ]
    ];

    CGRect rect = [ UIScreen fullScreenApplicationContentRect ];
    rect.origin.x = rect.origin.y = 0.0f;

    mainView = [ [ MainView alloc ] initWithFrame: rect ];

    [ window setContentView: mainView ];
    [ window orderFront: self ];
    [ window makeKey: self ];
    [ window _setHidden: NO ];
}
```

```
    NSTimer *timer = [ NSTimer scheduledTimerWithTimeInterval: 0.10
    target: self
    selector: @selector(handleTimer:)
    userInfo: nil
    repeats: YES ];
}

- (void) handleTimer: (NSTimer *) timer
{
    progress += 0.01;
    if (progress <= 1.0)
    {
        [ progressBar setProgress: progress ];
        [ progressBar updateIfNeeded ];
    }
}
@end

@implementation MainView
- (id) initWithFrame: (CGRect) rect {

    self = [ super initWithFrame: rect ];
    if (nil != self) {

        progressBar = [ [ UIProgressbar alloc ] initWithFrame: rect ];
        [ progressBar setStyle: 0 ];
        [ self addSubview: progressBar ];
    }
    return self;
}

- (void) dealloc
{
    [ self dealloc ];
    [ super dealloc ];
}
@end
```

Как это работает

Пример строки прогресса работает точно так же, как и пример индикатора прогресса:

1. При порождении приложения создается объект `MainView` и вызывается его метод `initWithFrame`.
2. Метод `initWithFrame` создает объект `UIProgressbar` и прикрепляет его к основному виду.
3. Создается объект `NSTimer` с триггером в 0,1 секунды, повторяющийся бесконечно. Каждый цикл триггера получает уведомление метод `handleTimer`, который увеличивает значение строки прогресса на 1% (0,01).

Progress HUDs: когда важно блокировать любые действия

Существует еще один объект определения прогресса, но он сам по себе не является индикатором. Класс `UIProgressHUD` отображает объект `UIProgressIndicator` и любой сопутствующий текст в полупрозрачном окне. Когда вы хотите донести до пользователя информацию о том, что он действительно не должен что-либо делать в каком-либо определенном окне до момента завершения операции, то именно для этих целей и существует класс `UIProgressHUD`. Такое окно располагается поверх окна целого вида, затемняя и надежно блокируя доступ к любым другим компонентам данного вида. Это можно увидеть в iPhone в случаях, когда меняются определенные возможности транспортного уровня, а также когда блокируется доступ к клавиатуре при отправке сообщений SMS.

Чтобы создать объект `UIProgressHUD`, инициализируйте его вместе с областью отображения родительского объекта:

```
UIProgressHUD *hud = [ [ UIProgressHUD alloc ] initWithFrame: viewRect ];
```

Текст для HUD задается с помощью метода `setText` этого объекта:

```
[ hud setText: @"Please Wait. I'm doing something REALLY important." ];
```

Затем прикрепите данный HUD к большому виду или окну:

```
[ mainView addSubview: hud ];
```

Хотя этот объект и инкапсулирует `UIProgressIndicator`, однако ни метод `startAnimation`, ни метод `stopAnimation` напрямую недоступны. Индикатор HUD активируется и деактивируется вызовом метода `show`:

```
[ hud show: YES ];
```

Когда придет время отобразиться, HUD затенит родительский вид. Затем он активирует вращающийся индикатор и отобразит текст, который был для него определен. Когда ваше приложение завершит обработку, спрячьте HUD:

```
[ hud show: NO ];
```

Пример: "Hello, HUD!"

Пример в этом разделе берет приложение "Hello, World!" из главы 3 и добавляет к нему пятисекундный `UIProgressHUD`. Этот HUD будет отображаться поверх объекта `UITextView`, а затем удаляться после истечения времени таймера.

Чтобы скомпилировать данный пример из командной строки, воспользуйтесь пакетом инструментов следующим образом:

```
$ arm-apple-darwin9-gcc -o MyExample MyExample.m -lobjc \
  -framework UIKit -framework Foundation -framework CoreFoundation
```

Листинги 7.9 и 7.10 содержат соответствующий код.

Листинг 7.9. Пример HUD прогресса (MyExample.h)

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

@interface MainView : UIView
{
    UITextView *textView;
}

- (id)initWithFrame:(CGRect)frame;
- (void)dealloc;
@end
```

```
@interface MyApp : UIApplication
{
    UIWindow *window;
    MainView *mainView;
}

- (void)applicationDidFinishLaunching:(NSNotification *)aNotification;
@end
```

Листинг 7.10. Пример HUD прогресса (MyExample.m)

```
#import "MyExample.h"

int main(int argc, char **argv)
{
    return UIApplicationMain(argc, argv, @"MyApp", @"MyApp");
}

UIProgressHUD *hud;

@implementation MyApp

- (void)applicationDidFinishLaunching:(NSNotification *)aNotification {
    window = [ [ UIWindow alloc ] initWithContentRect:
        [ UIScreen fullScreenApplicationContentRect ]
    ];
    CGRect rect = [ UIScreen fullScreenApplicationContentRect ];
    rect.origin.x = rect.origin.y = 0.0f;

    mainView = [ [ MainView alloc ] initWithFrame: rect ];

    [ window setContentView: mainView ];
    [ window orderFront: self ];
    [ window makeKey: self ];
    [ window _setHidden: NO ];
}
```

```
hud = [ [ UIProgressHUD alloc ] initWithFrame: rect ];
[ hud setText: @"Please Wait" ];
[ mainScreen addSubview: hud ];
[ hud show: YES ];

NSTimer *timer = [ NSTimer scheduledTimerWithTimeInterval: 5.0
target: self
selector: @selector(handleTimer:)
userInfo: nil
repeats: NO ];
}

- (void) handleTimer: (NSTimer *) timer
{
    [ hud show: NO ];
}
@end

@implementation MainView
- (id) initWithFrame: (CGRect) rect {
    self = [ super initWithFrame: rect ];
    if (nil != self) {
        textView = [ [ UITextView alloc ] initWithFrame: rect ];
        [ textView setText: @"Hello, World!" ];
        [ self addSubview: textView ];
    }
    return self;
}

- (void) dealloc
{
    [ self dealloc ];
    [ super dealloc ];
}
@end
```



Как это работает

HUD прогресса работает очень схожим образом, что и индикаторы, рассмотренные ранее в этой главе:

1. При порождении приложения создается объект `MainView` и вызывается его метод `initWithFrame`.
2. Метод `initWithFrame` создает объект `UITextView` и прикрепляет его к основному виду.
3. После того как основной вид добавлен к окну, создается `UIProgressHUD` и прикрепляется к нему, а затем активизируется.
4. Создается объект `NSTimer` с триггером в 5 секунд. Каждый цикл триггера `UIProgressHUD` получает указание скрыться, открывая под собой исходный `UITextView`.

Для дальнейшего изучения

- Воспользуйтесь своими знаниями листов предупреждений, полученными в *главе 3*, для создания "листа прогресса" без кнопок. Для заполнения строки используйте объект `NSTimer`. Когда строка достигнет своего края, автоматически уберите лист предупреждений. Такой сценарий идеально подойдет для тех случаев, когда вашему приложению требуется проверить наличие каких-либо обновлений продукта в Интернете.
- Проверьте наличие прототипов `UIProgressbar.h`, `UIProgressHUD.h` и `UIProgressIndicator.h` в папке пакета инструментов. Они могут находиться в папке `includes` вашего пакета инструментов: `/toolchain/sys/usr/include/UIKit`.

Обработка изображений

Платформа `UIKit` существенно облегчает работу с изображениями. Представленный в ней набор классов обработки изображений позволяет вам отображать, масштабировать, отсекают и накладывать изображения, создавая тем самым необходимые эффекты в вашем приложении. Вместо того чтобы раздувать один класс изображений невнятными процедурами, разработчики Apple выбрали более мудрый путь и наследовали каждый тип преобразования изображений, каждый из которых в свою очередь предлагает всего лишь несколько различных методов.

Объект изображения

Класс `UIImage` инкапсулирует само изображение. Он может использоваться для прорисовки напрямую внутри вида или выступать в роли контейнера объекта в более мощных классах видов изображений. Данный класс предоставляет методы для загрузки изображения из различных источников, задания ориентации изображения на экране и предоставления информации об этом изображении. Для простой графики `UIImage` может использоваться в процедурах прорисовки вида для визуализации изображений и узоров. Это является своего рода посредническими услугами между более крупными, полными классами видов изображений и работой с низкоуровневыми графическими процедурами, такими как буферы Core Surface.

Объект `UIImage` может быть создан из файла, взят из копии буфера Core Surface, или даже импортирован из фонового рисунка рабочего стола. Существуют как статические методы, так и экземпляры методов; поэтому на изображения можно ссылаться, их можно кэшировать или порождать экземпляры объектов новых изображений в зависимости от потребностей вашего приложения.

Статические методы

Самый простой способ сослаться на изображение — это воспользоваться статическими методами `UIImage`. Вместо того чтобы управлять экземплярами изображений, статические методы предоставляют прямой интерфейс к общим объектам, расположенным во внутреннем кэше памяти оболочки. Это помогает уменьшить беспорядок в вашем приложении и устраняет необходимость в проведении чистки. Статические методы существуют для получения доступа к изображениям напрямую из вашей программной папки, используя для этого путь файла, или из фонового рисунка рабочего стола.

□ `applicationImageNamed` — метод, который является наиболее предпочтительным для использования с изображениями, относящимися к приложению, поскольку всегда существует вероятность того, что пользователь переименует ваше приложение или установит его в другую папку, разрушив тем самым прямые пути к файлам. Чтобы использовать этот метод, укажите только имя файла. Платформа сама выяснит, куда установлено приложение, и загрузит нужное изображение:

```
UIImage *image = [ UIImage applicationImageNamed: @"image.png" ];
```

- `imageAtPath` — если нужное изображение находится вне папки вашего приложения (например, изображения, сделанные фотоаппаратом), то сослаться на это изображение можно с помощью прямого пути:

```
UIImage *image = [ UIImage imageAtPath: @"/path/to/image.png" ];
```

- `defaultDesktopImage` — метод, применяемый для возврата ссылки на фоновый рисунок рабочего стола, который задается в приложении **Settings**. Мы активно его используем в нашем примере, поскольку он освобождает вас от необходимости загружать изображения для использования в примерах:

```
UIImage *image = [ UIImage defaultDesktopImage ];
```

Экземпляры методов

Помимо статических ссылок изображения также могут быть подвергнуты обработке как объекты, размещенные в вашем приложении.

- `initWithContentsOfFile`. Наиболее распространенный подход — это предоставить прямой путь к файлу изображения:

```
UIImage *image = [ [ UIImage alloc ] initWithContentsOfFile:
    @"/path/to/image.png"
];
```

Данный метод может быть также вызван с аргументом кэша, указывая тем самым оболочке кэшировать изображение при его загрузке так, чтобы его не нужно было многократно считывать с диска:

```
UIImage *image = [ [ UIImage alloc ]
    initWithContentsOfFile: @"/path/to/image.png"
    cache: YES
];
```

- `initWithCoreSurfaceBuffer`. Если ваше приложение использует платформу `Core Surface` для создания видеобуфера `CoreSurfaceBuffer` (рассмотренного в главе 5), то объект `UIImage` может быть создан как мгновенная копия текущего буфера:

```
UIImage *image = [ [ UIImage alloc ]
    initWithCoreSurfaceBuffer: screenSurface
];
```

Отображение изображения

Объекты видов имеют внутренние процедуры прорисовки, которые вызываются, когда вызываются их методы `drawRect`. В отличие от других классов изображений `UIImage` не может быть прикреплен напрямую к объекту вида как подвид. Вместо этого классы, порожденные от `UIView`, могут подменять их методы `drawRect` для включения вызова методов прорисовки объекта изображения.

Метод `drawRect` объекта вида вызывается всякий раз, когда необходимо визуализировать часть его окна. Чтобы визуализировать содержимое `UIImage` внутри этого окна, вызовите метод `draw1PartImageInRect` объекта:

```
- (void)drawRect:(CGRect)rect {
    CGRect myRect;
    CGSize imageSize = [ image size ];

    myRect.origin.x = 0;
    myRect.origin.y = 0;
    myRect.size.width = imageSize.width;
    myRect.size.height = imageSize.height;
    [ image draw1PartImageInRect: myRect ];
}
```

Будьте осторожны и не размещайте какие-либо новые объекты внутри метода `drawRect`, поскольку он вызывается каждый раз, когда необходимо перерисовать окно.

Вывод на экран узоров

Если изображение является узором, то оно может повторяться в пределах всей области вида с помощью другого метода, предоставляемого в классе `UIImage` — `drawAsPatternInRect`:

```
[ image drawAsPatternInRect: rect ];
```

Этот метод повторяет указанное изображение в пределах отображаемого фрейма.

Пример: развлечение со значками

Данный пример иллюстрирует вывод на экран изображений и узоров в рамках метода `drawRect` класса вида. Мы создаем пустой класс вида, а затем

подменяем `drawRect` для включения процедур прорисовки, отображая в основном окне значки некоторых приложений.

Чтобы скомпилировать данный пример из командной строки, воспользуйтесь пакетом инструментов следующим образом:

```
$ arm-apple-darwin9-gcc -o MyExample MyExample.m -lobjc \
    -framework CoreFoundation -framework Foundation -framework UIKit
```

Листинги 7.11 и 7.12 содержат соответствующий код.

Листинги 7.11. Пример использования UIImage (MyExample.h)

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

@interface MainView : UIView
{
}
- (void)drawRect:(CGRect)rect;
@end

@interface MyApp : UIApplication
{
    UIWindow *window;
    MainView *mainView;
}
- (void)applicationDidFinishLaunching:(NSNotification *)aNotification;
@end
```

Листинги 7.12. Пример использования UIImage (MyExample.m)

```
#import <Foundation/Foundation.h>
#import <CoreFoundation/CoreFoundation.h>
#import "MyExample.h"

int main(int argc, char **argv)
{
    return UIApplicationMain(argc, argv, @"MyApp", @"MyApp");
}
```

```
@implementation MyApp
- (void)applicationDidFinishLaunching:(NSNotification *)aNotification {
    window = [ [ UIWindow alloc ] initWithContentRect:
        [ UIScreen fullScreenApplicationContentRect ]
    ];

    CGRect rect = [ UIScreen fullScreenApplicationContentRect ];
    rect.origin.x = rect.origin.y = 0.0f;

    mainView = [ [ MainView alloc ] initWithFrame: rect ];
    [ window setContentView: mainView ];
    [ window orderFront:.self ];
    [ window makeKey: self ];
    [ window _setHidden: NO ];

}
@end

@implementation MainView
- (void)drawRect:(CGRect)rect {
    CGRect drawRect;
    CGSize size;

    UIImage *pattern = [ UIImage imageAtPath:
        @"/Applications/MobilePhone.app/icon.png" ];
    [ pattern drawAsPatternInRect: rect ];

    UIImage *image = [ UIImage imageAtPath:
        @"/Applications/MobileSafari.app/icon.png" ];
    size = [ image size ];
    drawRect.origin.x = (320 - (size.width)) / 2;
    drawRect.origin.y = (480 - (size.height)) / 2;
    drawRect.size.width = size.width;
    drawRect.size.height = size.height;
    [ image draw1PartImageInRect: drawRect ];
}
@end
```

UIImageView: вид с видом

Класс `UIImageView` предоставляет способ для работы с изображением как с элементами управления. Это оказывается весьма кстати, когда изображение необходимо связать с объектами вида, панелями инструментов или ячейками таблицы, или же для таких приложений, как показы слайдов, где вся область вида может содержать изображение.

Объект `UIImageView` выступает в роли обертки вида для `UIImage`; т. е. сначала объект `UIImage` создается, а затем прикрепляется к объекту `UIImageView` с помощью метода `initWithImage` или метода `setImage` класса:

```
UIImage *image = [ UIImage imageAtPath: @"/path/to/image.png" ];
UIImageView *imageView = [ [ UIImageView alloc ]
    initWithImage: image
];
```

Координаты инициализируются, когда фрейм вида определяет смещение относительно родительского вида, в котором будет прорисовываться изображение:

```
CGRect rect = CGRectMake(0, 0, 320, 200);
UIImageView *imageView = [ [ UIImageView alloc ]
    initWithRect: rect ];
[ imageView setImage: [ UIImage imageAtPath:
    @"/path/to/image.png" ] ];
```

После своего создания изображение может быть прикреплено к объекту вида любого типа, ячейке таблицы или другому похожему объекту:

```
[ preferencesCell addSubview: imageView ];
```

Также можно перейти к использованию `UITransitionView`:

```
[ transitionView transition: 0 toView: imageView ];
```

Чтобы масштабировать изображение, необходимо всего лишь установить размер фрейма. Затем новый размер может быть применен с помощью метода `setFrame` класса:

```
rect.size.width = 160;
rect.size.height = 240;
[ imageView setFrame: rect ];
```

UIAutocorrectImageView: масштабирование

Класс `UIAutocorrectImageView` похож на класс `UIImageView`, за исключением того, что изображение автоматически масштабируется в соответствии с размером в его фрейме:

```
CGRect rect;  
rect.origin.x = 80;  
rect.origin.y = 120;  
rect.size.width = 160;  
rect.size.height = 240;  
  
UIAutocorrectImageView *imageView = [  
    [ objc_getClass("UIAutoCorrectImageView") alloc ]  
    initWithFrame: rect  
    image: [ UIImage defaultDesktopImage ]  
];
```

Данный пример создает вид изображения с помощью фонового рисунка рабочего стола. Оно масштабируется до 160×240 и отображается со смещением в 80×120 в пределах того класса вида, к которому оно прикреплено.

После того как изображение будет создано, оно может быть прикреплено к объекту вида любого типа, точно так же, как и `UIImageView`:

```
[ mainView addSubview: imageView ];
```

Изображение масштабируется в соответствии с шириной и высотой, заданной во фрейме, точно так же, как и класс `UIImageView`. Геометрическое соотношение может быть изменено, поэтому, масштабируя размер фрейма, будьте осторожны и сохраните его.

UIClippedImageView: обрезка кругов — квадраты

Как и класс `UIAutocorrectImageView`, класс `UIClippedImageView` позволяет определять меньшую область отображения. Вместо того чтобы масштабировать изображение, чтобы оно поместилось во фрейме, этот класс обрезает изображение, отображая только часть изображения в области вида:

```
CGRect rect;  
rect.size.width = 160;
```

```
rect.size.height = 240;
rect.origin.x = 80;
rect.origin.y = 120;

UIClippedImageView *imageView = [ [ objc_getclass("UIClippedImageView")
alloc ]
initWithFrame: rect
image: [ UIImage defaultDesktopImage ]
];
```

Данный пример создает обрезанное изображение во фрейме вида 160×240, расположенном посередине экрана. Это означает, что отображаться будет только 160×240 всего изображения, а все остальное будет обрезано:

```
CGPoint origin;

origin.x = 0;
origin.y = 0;
[ self setImageOrigin: origin ];
```

Чтобы отсечь другую часть изображения, можно поменять начало координат изображения. В этом примере начало координат установлено в (0, 0), в результате чего окно 160×240 незамедлительно переместится в левый верхний угол изображения. Поскольку это можно поменять в процессе отображения изображения, то с этим классом можно проделать несколько ловких манипуляций, как будет показано в следующем примере.

Когда изображение будет готово к выводу на экран, оно может быть добавлено к существующему объекту вида:

```
[ mainView addSubview: imageView ];
```

UICompositeImageView: **многоуровневая прозрачность**

Apple славится своими великолепными перекрытиями и прозрачностью. Один из способов сделать это возможным и на iPhone — использование класса `UICompositeImageView`. Этот класс позволяет нескольким изображениям накладываться друг на друга, применяя разные уровни прозрачности, а каждому изображению просвечивать насквозь. Такой эффект отлично подхо-

дит для разработки собственных пользовательских интерфейсов, например, экранов видеоигр, имеющих перекрытия элементов управления.

Данный класс использует простой интерфейс, позволяя изображениям добавляться по очереди. Изображения могут накладываться поверх друг друга или добавляться в различные части окна:

```
UICompositeImageView *compositeView = [
    [ objc_getClass("UICompositeImageView") alloc ] initWithFrame: rect
];
```

После того как изображения будут созданы, они по отдельности добавляются путем вызова метода `addImage:`

```
[ compositeView addImage: [ UIImage defaultDesktopImage ] ];
```

Прозрачность

Затененность каждого изображения можно настроить при его создании (в программе рисования) или же при добавлении его к составному виду во время исполнения. Составной вид сам по себе может выполнять над уровнем нового изображения после его добавления ограниченное количество операций. Они включают в себя операции, перечисленные в табл. 7.2.

Таблица 7.2

Операция	Описание
1	Установка яркости (intensity)
2	Установка затененности (opacity)

Чтобы сделать изображение полупрозрачным, используйте операцию затененности (2). Аргумент `fraction` задает уровень затененности изображения, в следующем примере — 50%:

```
[ compositeView addImage:
    [ UIImage imageNamed: @"overlay.png" ]
    operation: 2
    fraction: 0.5
];
```

Масштабирование и размещение

Когда изображение добавляется к составному виду, его можно масштабировать и/или поместить поверх других уровней несколькими способами. `UICompositeImageView` поддерживает две дополнительные версии метода `addImage`, каждая из которых принимает исходный и целевой фрейм для масштабирования и размещения. Объект изображения сравнивает эти два фрейма и осуществляет те изменения, которые вызваны выявленными различиями:

```
CGRect src, dest;
src.origin.x = 100;
src.origin.y = 50;
src.size.width = 320;
src.size.height = 480;
dest.origin.x = 80;
dest.origin.y = 120;
dest.size.width = 160;
dest.size.height = 120;
```

```
[ compositeView addImage:
 [ UIImage imageNamed: @"overlay.png" ]
 toRect: dest
 fromRect: src
];
```

Данный пример создает два фрейма: `src` и `dest`. Структура `src` содержит начальный размер изображения (320×480) и координаты для использования в рамках исходного изображения (100×50). Все, что находится левее или выше этих координат, будет обрезано. Координаты, предоставленные структурой `dest`, обозначают левый верхний угол составного вида, в который будет вставлено новое изображение.

Указывая различные размеры фреймов между структурами `src` и `dest`, вы можете добиться автоматического изменения размеров изображения.

Для выполнения операции, описанной в предыдущем разделе, наряду с масштабированием и размещением в одном методе, существует другая версия `addImage`. Она объединяет все четыре входных аргумента, позволяя вам мас-

штабировать и размещать новое изображение одновременно с изменением его затененности и яркости:

```
[ compositeView addImage:
  [ UIImage imageNamed: @"overlay.png" ]
  toRect: dest
  fromRect: src
  operation: 2
  fraction: 0.35
];
```

Приведенный пример загружает изображение `overlay.png` из программной папки приложения. Структуры `CGRect` указывают составному виду вставить изображение, начиная с точки (100, 50), в составной вид в точке координат (80, 20). Кроме того, составному виду указывается сделать новый уровень непрозрачным на 35%.

К составному изображению можно добавить любое количество уровней. После того как составное изображение будет сформировано, оно может быть добавлено к существующему объекту `UIView`:

```
[ mainScreen addSubview: compositeView ];
```

Пример: интересная анимация обрезки

Класс `UIClippedImageView` — весьма интересный для работы класс, поскольку он позволяет вам осуществлять некоторые эффекты с помощью механизма обрезки. Далее в примере мы создадим `UIClippedImageView` и используем таймер для непрерывной прокрутки обрезанной области. Тем самым будет создан эффект перемещающегося окна, открывающего различные части изображения.

Чтобы скомпилировать данный пример из командной строки, воспользуйтесь пакетом инструментов следующим образом:

```
$ arm-apple-darwin9-gcc -o MyExample MyExample.m -lobjc \
  framework UIKit -framework CoreFoundation -framework Foundation
```

В листингах 7.13 и 7.14 содержится соответствующий код.

Листинг 7.13. Пример обрезки (MyExample.h)

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

@interface MainView : UIView
{
}
- (id)initWithFrame:(CGRect)rect;
- (void)dealloc;
@end

@interface MyApp : UIApplication
{
    UIWindow *window;
    MainView *mainView;
}

- (void)applicationDidFinishLaunching:(NSNotification *)aNotification;
@end
```

Листинг 7.14. Пример обрезки (MyExample.m)

```
#import <Foundation/Foundation.h>
#import <CoreFoundation/CoreFoundation.h>
#import <GraphicsServices/GraphicsServices.h>
#import <UIKit/CDStructures.h>
#import <UIKit/UIKit.h>
#import <UIKit/UIClippedImageView.h>
#import "MyExample.h"

CGPoint origin;
UIClippedImageView *imageView;

int main(int argc, char **argv)
{
```

```
return UIApplicationMain(argc, argv, @"MyApp", @"MyApp");
}

@implementation MyApp

- (void)applicationDidFinishLaunching:(NSNotification *)aNotification {
    window = [ [ UIWindow alloc ] initWithContentRect:
        [ UIHardware fullScreenApplicationContentRect ]
    ];

    CGRect rect = [ UIHardware fullScreenApplicationContentRect ];
    rect.origin.x = rect.origin.y = 0.0f;

    mainView = [ [ MainView alloc ] initWithFrame: rect ];
    [ window setContentView: mainView ];
    [ window orderFront: self ];
    [ window makeKey: self ];
    [ window _setHidden: NO ];

    NSTimer *timer = [ NSTimer scheduledTimerWithTimeInterval: 0.10
        target: self
        selector: @selector(handleTimer:)
        userInfo: nil
        repeats: YES ];
}

- (void) handleTimer: (NSTimer *) timer
{
    if (origin.y > 0)
        origin.y --;
    else
        origin.y = 480;
    if (origin.x > 0)
        origin.x --;
    else
```

```
        origin.x = 320;
        [ imageView setOriginAdjustingImage: origin ];
    }
@end

@implementation MainView
- (id)initWithFrame:(CGRect)rect {
    self = [ super initWithFrame: rect ];
    if (nil != self) {
        origin.x = 80;
        origin.y = 120;

        rect.size.width = 160;
        rect.size.height = 240;
        rect.origin.x = 80;
        rect.origin.y = 120;

        imageView = [ [ objc_getClass("UIClippedImageView") alloc ]
            initWithFrame: rect
            image: [ UIImage defaultDesktopImage ]
        ];
        [ imageView setImageOrigin: origin ];
        [ self addSubview: imageView ];
    }
    return self;
}

- (void)dealloc
{
    [ self dealloc ];
    [ super dealloc ];
}
@end
```

Как это работает

1. При порождении приложения создается экземпляр класса `MainView` и вызывается его метод `initWithFrame`.
2. Метод `initWithFrame` создает объект `UIClippedImageView` с использованием области вида посередине экрана и размером отображения 160×240 .
3. Создается объект `NSTimer`, который каждую $0,1$ часть секунды уведомляет метод `handleTimer`. Эта подпроцедура смещает начало координат изображения на один пиксел вверх и один пиксел влево, заворачивая его при необходимости.
4. После того как будет установлено начало координат, видимое окно изображения перемещается к новым координатам, создавая ощущение скользящего по картинке окна.

Для дальнейшего изучения

- Измените пример `UIImage` из этого раздела, чтобы случайным образом отображать различные значки приложений в разных местах экрана.
- Измените пример `UIClippedImageView` из этого раздела, чтобы сместить размер области отображения с помощью метода `setFrame`.
- Проверьте наличие прототипов `UIImage.h`, `UIImageView.h`, `UIAutocorrectImageView.h` и `UIClippedImageView.h` в папке `include` вашего пакета инструментов. Их можно найти в папке `/toolchain/sys/include/UIKit`.

Списки разделов

Об объекте `UITableView` вы узнали в *главе 3* как о средстве отображения списков выбора информации. Когда таблица становится достаточно большой, то нахождение в ней какого-либо определенного элемента становится сродни поиску иголки в стоге сена. Класс `UISectionList` предоставляет структуру, аналогичную `UITableView`, но расширенную и включающую в себя отдельные группировки ячеек и панель прокрутки типа Rolodex для быстрого перехода к заголовку раздела. Каждой группировке может быть сопоставлен заголовок раздела наподобие жанра книг или первой буквы контакта. Списки разделов (section lists) применяются в собственных контактах iPhone и списках песен.

Класс `UISectionList` инкапсулирует `UISectionTable`, который включает в себе табличную часть списка. Как и другие таблицы, список разделов использует привязку данных (`data binding`). Привязка данных — это интерфейс к источнику данных, к которому обращается таблица за содержимым и структурой таблицы. Источник данных для списка разделов предоставляет методы обратного вызова, необходимые для построения группировок списка разделов, заголовков разделов и отдельных ячеек строк.

Как и в случае с другими табличными классами, описываемыми в этой книге, приводимые здесь примеры создают подкласс объекта `UISectionList`, который может выступать в роли списка разделов и его собственного источника данных. Такая архитектура является наиболее наглядной и лучше всего подходит для создания специализированных повторно используемых классов.

Создание списка разделов

Чтобы создать список разделов, который помимо всего сможет выступать в роли собственного источника данных, унаследуйте `UISectionList`. Приведенный далее код создает класс `MySectionList`, содержащий методы как для списка разделов, так и для его источника данных:

```
@interface MySectionList : UISectionList
{
    UISectionTable *table;
}

/* Методы списка разделов */
- (id)initWithFrame:(CGRect)rect;
- (void)dealloc;

/* Методы источника данных */
- (int)numberOfSectionsInSectionList:(UISectionList *)aSectionList;
- (NSString *)sectionList:
    (UISectionList *)aSectionList
    titleForSection:(int)section;
- (int)sectionList:
    (UISectionList *)aSectionList
    rowForSection:(int)section;
```

```
- (int) numberOfRowsInTable: (UITableView *)table;
- (UITableViewCell *) table: (UITableView *)table
  cellForRow: (int)row
  column: (int)col;
@end
```

На объект `UISectionList` работают следующие методы источника данных:

- `NumberOfSectionsInSectionList` — возвращает количество разделов, т. е. количество заголовков различных категорий, в списке разделов. Например, если ваше приложение было бы менеджером списка контактов, то вы бы определили 26 разделов (по одному на каждую букву). Пустые разделы должны быть выброшены из списка разделов, уменьшив тем самым это количество;
- `titleForSection` — каждый раздел имеет заголовок, даже если он состоит всего лишь из одной буквы. Если бы ваше приложение было книжным магазином, то заголовки ваших разделов могли бы включать такие названия, как "Computer Science" и "Reference". Метод `titleForSection` вызывается тогда, когда требуется название для заданного раздела. Он предоставляет в качестве аргумента порядковый номер раздела и возвращает объект `NSString`, содержащий соответствующее название;
- `rowForSection` — каждая строка в списке разделов помечается номером. Строка должна быть связана с разделом, которому она принадлежит, так чтобы она могла отображаться под корректным заголовком раздела. Список разделов исходит из того, что в рамках массива строк вашего приложения каждый раздел имеет последовательный ряд строк.

Метод `rowForSection` запрашивает номер начальной строки вашего списка строк в соответствии с заданным разделом. Например, если во всей вашей таблице имеется 10 строк, то первый раздел может начинаться со строки 0, а второй раздел — со строки 7. Список разделов будет понимать это как то, что строки с 0 по 6 соответствуют первому разделу, а строки с 7 по 10 — второму;

- `numberOfRowsInTable` — этот метод возвращает полное количество строк в данном списке разделов по всем разделам;
- `cellForRow` — отдельные строки инкапсулируются объектами `UITableViewCell`. Этот объект уже обсуждался в *разд. "Таблицы"* ранее в этой главе. Поскольку строки в списке разделов относятся к видимому фрейму экрана, то данный метод запрашивает возвращение ячейки заданной строки.

Инициализация

Чтобы построить полнофункциональный список разделов, необходимо запрограммировать все описанные в предыдущем разделе методы. Мы покажем один из вариантов их написания в нашем примере в конце этого раздела.

Само по себе создание списка разделов осуществляется путем вызова класса вида или класса приложения. Список может быть задан как содержимое всего окна или же создан как вид, который позднее перемещается управляющим видом. Приведенный далее код размещает объект `MySectionList` и инициализирует его:

```
UISectionList *sectionList = [ [ MySectionList alloc ]
    initWithFrame: windowRect ];
```

Как и в случае с классом `UITable`, для инициализации источника данных должен быть вызван метод `reloadData`. Чтобы загрузить информацию для отображения в строках, разделах и заголовках разделов списка, этот метод может быть подменен вашим подклассом:

```
[ sectionList reloadData ];
```

Получение доступа к объекту таблицы

Список разделов представляет свои данные путем инкапсулирования объекта `UISectionTable`. Этот класс порожден от базового класса `UITable` и ведет себя схожим образом. Сама таблица должна также быть инициализирована так, чтобы иметь для отображения информации, по крайней мере, один столбец таблицы. Список разделов содержит метод `table`, который возвращает указатель на объект `UISectionTable`. Этот объект был создан внутренне вместе с объектом списка разделов.

Приведенный далее код в методе `initWithFrame` объекта `MySectionList` предназначен для инициализации табличной части списка:

```
UISectionTable *table = [ self table ];
UITableColumn * column = [ [ UITableColumn alloc ]
    initWithTitle:@"Column Name"
    identifier:@"column"
    width: 320.0 ];
[ table addTableColumn: column ];
[ table setSeparatorStyle: 1 ];
```

```
[ table setRowHeight: 48.0 ];  
[ table setDelegate: self ];
```

Методы, использованные для конфигурации табличной части списка разделов, идентичны объекту `UITableView`, описанному в *главе 3*. Здесь должен быть создан, по крайней мере, один столбец и заданы различные атрибуты таблицы.

Отображение списка разделов

Список разделов может быть отображен путем задания его в качестве содержимого окна, добавления его как подвида к существующему виду или перехода его к использованию класса `UITransitionView`, описанного в *главе 3*.

Чтобы задать список разделов в качестве содержимого окна, воспользуйтесь `setContentViewController:`

```
[ window setContentViewController: sectionList ];
```

Чтобы добавить список разделов как подвид, вызовите `addSubview:`

```
[ self addSubview: sectionList ];
```

Чтобы использовать переходной вид для перехода к списку разделов, воспользуйтесь следующим кодом:

```
[ transitionView transition: 1 toView: sectionList ];
```

Как только список разделов будет отображен, строки, попавшие в зону видимости, станут толчком для списка запросить метод `cellForRow:`. Этот метод возвращает объект `UITableViewCell` для запрошенной строки. Более подробную информацию о классе `UITableViewCell` см. в разд. "Таблицы" ранее в этой главе.

События выбора

Когда пользователь в списке разделов выбирает какой-либо элемент, вызывается метод `tableView:didSelectRowAtIndexPath:`. Затем этот метод выполняет все необходимые приложению операции.

Встроенный табличный метод `indexPathForSelectedRow` возвращает количество строк, выбранных пользователем:

```
- (void)tableView:(UITableView *)tableView didSelectRowAtIndexPath {  
    int rowSelected = [ tableView indexPathForSelectedRow ].rowIndex;  
    /* Делайте здесь что-нибудь */  
}
```

Пример: выбор файлов

Одним весьма полезным вариантом использования списка разделов является возможность организации выбора файлов для открытия документов, приложений или относящихся к приложению файлов. Рассматриваемый далее пример определяет класс `FileSelector`, порожденный от `UITableView`, который отображает список файлов, имеющих заданное расширение и расположенных в заданной папке. Заголовками разделов списка разделов являются первые буквы имен файлов.

Чтобы воспользоваться этим классом, создайте объект `FileSelector`, потом передайте путь и расширение, которые вы хотите использовать. Затем вызовите метод `reloadData`, в результате чего просканируется указанная папка и создастся список разделов:

```
fileSelector = [ [ FileSelector alloc ]
    initWithFrame: rect ];
[ fileSelector setPath: @"/Applications" ];
[ fileSelector setExtension: @".app" ];
[ fileSelector reloadData ];
```

После того как список разделов будет построен, он может быть добавлен к существующему виду или задан как содержимое окна. Например:

```
[ mainWindow addSubview: fileSelector ];
```

Источник данных поддерживает несколько различных массивов:

- `files` — список объектов `UITableViewCell`, соответствующих каждой строке в списке разделов;
- `filenames` — действительные имена файлов каждой строки;
- `sections` — список названий разделов, который построен в ходе перечисляющего цикла при сканировании файлов в папке. В качестве заголовков разделов используются первые буквы имен файлов;
- `offsets` — список смещений строк, связанных с группировками разделов среди всех четырех массивов. Например, раздел 1 может начинаться в строке 5.

Чтобы скомпилировать данный пример из командной строки, воспользуйтесь пакетом инструментов следующим образом:

```
$ arm-apple-darwin9-gcc -o MyExample MyExample.m -lobjc \
    -framework UIKit -framework CoreFoundation -framework Foundation
```

Листинги 7.15 и 7.16 содержат соответствующий код для таблицы выбора файлов, а в листингах 7.17 и 7.18 приведено само приложение.

Листинг 7.15. Пример выбора файлов (FileSelector.h)

```
#import <CoreFoundation/CoreFoundation.h>
#import <Foundation/Foundation.h>
#import <UIKit/UIKit.h>
#import <UIKit/UISectionList.h>
#import <UIKit/UITableView.h>
#import <UIKit/UIImageAndTextTableCell.h>

@interface FileSelector : UISectionList
{
    UITableView *table;
    NSMutableArray *files;
    NSMutableArray *filenames;
    NSMutableArray *sections;
    NSMutableArray *offsets;
    NSString *path;
    NSString *extension;
}

/* Методы FileSelector */
- (id)initWithFrame:(CGRect)rect;
- (void)reloadData;
- (void)setPath:(NSString *)_path;
- (void)setExtension:(NSString *)_extension;
- (void)dealloc;

/* Методы источника данных */
- (int)numberOfSectionsInSectionList:(UISectionList *)aSectionList;
- (NSString *)sectionList:(UISectionList *)aSectionList
  titleForSection:(int)section;
- (int)sectionList:(UISectionList *)aSectionList
  rowForSection:(int)section;
- (int)numberOfRowsInTable:(UITableView *)table;
```

```
- (UITableViewCell *) table: (UITableView *)table cellForRowAtIndexPath:
    (int)row column:
    (int)col;
@end
```

Листинг 7.16. Пример выбора файлов (FileSelector.m)

```
#import "FileSelector.h"

@implementation FileSelector
- (id)initWithFrame:(CGRect)rect {
    self = [ super initWithFrame: rect ];
    if (nil != self) {
        path = nil;
        extension = nil;
        files = [ [ NSMutableArray alloc ] init ];
        sections = [ [ NSMutableArray alloc ] init ];
        offsets = [ [ NSMutableArray alloc ] init ];
        filenames = [ [ NSMutableArray alloc ] init ];

        [ self setShouldHideHeaderInShortLists: NO ];
        [ self setDataSource: self ];
    }
    return self;
}
- (void)reloadData {
    NSString *file;
    NSDirectoryEnumerator *dirEnum;
    char cFileName[256], lastSection[2], mySection[2];

    if (path == nil || extension == nil) {
        return;
    }

    [ files removeAllObjects ];
    [ sections removeAllObjects ];
```

```
[ offsets removeAllObjects ];
[ filenames removeAllObjects ];

dirEnum = [ [ NSFileManager defaultManager ]
  enumeratorAtPath: path ];
while ((file = [ dirEnum nextObject ])) {
  if ([ file hasSuffix: extension ] == YES) {
    UIImageAndTextTableCell *cell = [
      [ UIImageAndTextTableCell alloc ] init
    ];

    [ cell setTitle: [ file substringToIndex: [ file length ] - 4 ]
    ];

    strcpy(cFileName,
      [ file cStringUsingEncoding: NSASCIIStringEncoding ],
      sizeof(cFileName));
    mySection[0] = toupper(cFileName[0]);
    mySection[1] = 0;
    if (mySection[0] >= '0' && mySection[0] <= '9') {
      mySection[0] = '0';
    }

    if ([ sections count ] > 0) {
      NSString *lastSectionName = [ sections objectAtIndex:
        [ sections count ] - 1
      ];

      strcpy(lastSection,
        [ lastSectionName
          cStringUsingEncoding: NSASCIIStringEncoding ],
        sizeof(lastSection));

      if (mySection[0] != lastSection[0]) {
        [ sections addObject: [
```

```
        [ NSString alloc ] initWithCString: mySection ]
    ];
    [ offsets addObject:
        [ NSNumber numberWithInt: [ files count ] + 1 ]
    ];
}

} else {
    [ sections addObject: [
        [ NSString alloc ] initWithCString: mySection ]
    ];
    [ offsets addObject: [ NSNumber numberWithInt:
        [ files count ] + 1 ]
    ];
}

[ files addObject: cell ];
[ filenames addObject: file ];
}
}

table = [ self table ];
UITableViewColumn * column = [ [ UITableViewColumn alloc ]
    initWithTitle:@"Filename"
    identifier:@"filename"
    width: 320.0 ];
[ table addTableColumn: column ];
[ table setSeparatorStyle: 1 ];
[ table setRowHeight: 48.0 ];
[ table setDelegate: self ];

[ super reloadData ];
}

- (void)setPath:(NSString *)_path {
    path = _path;
}
}
```

```
- (void)setExtension:(NSString *)_extension {
    extension = _extension;
}

- (void)dealloc
{
    [ self dealloc ];
    [ files dealloc ];
    [ filenames dealloc ];
    [ sections dealloc ];
    [ offsets dealloc ];
    [ super dealloc ];
}

- (int)numberOfSectionsInSectionList:(UISectionList *)aSectionList {
    return [ sections count ];
}

- (NSString *)sectionList:(UISectionList *)aSectionList
    titleForSection:(int)section
{
    return [ sections objectAtIndex: section ];
}

- (int)sectionList:(UISectionList *)aSectionList
    rowForSection:(int)section
{
    return ([ [ offsets objectAtIndex: section ] intValue ] - 1);
}

- (int) numberOfRowsInTable: (UITable *)table
{
    return [ files count ];
}
```

```
- (UITableViewCell *) table: (UITableView *)table cellForRowAtIndexPath: (int)row
    column: (int)col
{
    return [ files objectAtIndex: row ];
}

- (UITableViewCell *) table: (UITableView *)table cellForRowAtIndexPath:
    (int)row column:
    (int)col
    reusing: (BOOL) reusing
{
    return [ files objectAtIndex: row ];
}

- (void)tableViewSelected:(NSNotification *)notification {
    NSString *file = [ filenames objectAtIndex:
        [ table selectedRow ]
    ];
    printf("Selected: %s\n", [ file
        cStringUsingEncoding: NSASCIIStringEncoding ]);
}
@end
```

Листинг 7.17. Пример выбора файлов (MyExample.h)

```
#import <CoreFoundation/CoreFoundation.h>
#import <Foundation/Foundation.h>
#import <UIKit/UIKit.h>
#import "FileSelector.h"

@interface MyApp : UIApplication
{
    UIWindow *window;
}
}
```

```
- (void)applicationDidFinishLaunching:(NSNotification *)aNotification;
@end
```

Листинг 7.18. Пример выбора файлов (MyExample.m)

```
#import "MyExample.h"

int main(int argc, char **argv)
{
    return UIApplicationMain(argc, argv, @"MyApp", @"MyApp");
}

@implementation MyApp

- (void)applicationDidFinishLaunching:(NSNotification *)aNotification {
    UIWindow *window = [ [ UIWindow alloc ] initWithContentRect:
        [ UIHardware fullScreenApplicationContentRect ]
    ];

    CGRect rect = [ UIHardware fullScreenApplicationContentRect ];
    rect.origin.x = rect.origin.y = 0.0f;

    FileSelector *fileSelector = [ [ FileSelector alloc ]
        initWithFrame: rect
    ];
    [ fileSelector setPath: @"/Applications" ];
    [ fileSelector setExtension: @".app" ];
    [ fileSelector reloadData ];

    [ window setContentView: fileSelector ];
    [ window orderFront: self ];
    [ window makeKey: self ];
    [ window _setHidden: NO ];
}

@end
```

Как это работает

1. При порождении приложения оно создает объект `FileSelector` и вызывает метод `initWithFrame`. Тем самым создаются необходимые объекты для хранения строк списка разделов.
2. Вызываются методы `setPath` и `setExtension` для назначения папки и расширения файла, которые хочет использовать вызывающий. Затем вызывается метод `reloadData`, что приводит к сканированию папки и построению списка файлов, заголовков разделов и ячеек для каждой строки в списке разделов. Наконец, список разделов задается как содержимое окна.
3. Вызывается источник данных, являющийся замкнутым внутри класса выбора файлов, для получения информации о количестве разделов и строк в таблице. Запрашиваются названия разделов, подсчитывается количество строк в каждом разделе.
4. После отображения списка разделов попавшие в зону видимости строки приводят к вызову метода `cellForRow` источника данных. Этот метод возвращает ячейку, соответствующую запрошенной строке.
5. Когда пользователь выбирает какой-либо элемент, вызывается метод `tableViewSelected`. Он ищет имя файла в локальном индексе. С этого момента за выполнение необходимых действий начинает отвечать код разработчика.

Для дальнейшего изучения

- Измените этот пример, чтобы вместо более экстравагантного класса `UISectionList` использовать только класс `UISectionTable`.
- Проверьте наличие прототипов `UISectionList.h` и `UISectionTable.h` в папке `include` пакета инструментов. Их можно найти в папке `/toolchain/sys/usr/include/UIKit`.

Выборщики

Выборщики (pickers) — это щелкающие колесики для iPhone: большие вращающиеся диски, которые могут содержать любое количество различных вариантов. Выборщики используются вместо раскрывающихся меню для предоставления пользователю графически разнообразного интерфейса выбора. Являющийся близким родственником элемента управления класс

UIPickerView был создан как полный класс вида по причине его абсолютного размера на экране. Это позволяет использовать его практически везде, включая основной вид, или в связке с таблицами предпочтений.

Создание выборщика

Класс UIPickerView содержит объект UIPickerViewTable, который порожден от объекта UITableView. Как и остальные таблицы, UIPickerViewTable использует привязку данных (data binding). В отличие от других табличных классов источник данных выборщика не задается методом dataSource. Вместо этого для получения запросов на привязку данных помимо событий выборщика используется делегат. Скорее всего, это было сделано для того, чтобы придать классу UIPickerView простоту элемента управления. Привязка данных UIPickerView достаточно мала, чтобы хорошенько спрятаться в управляющем виде, хотя сам выборщик также может быть унаследован для создания замкнутого выборщика.

При создании вида выборщика используйте окно высотой в 200 пикселей. Инициализация выборщика с любым другим размером фрейма приведет к игнорированию заданного размера. Выборщик может быть помещен в любом месте другого вида, но обычно располагается внизу:

```
UIPickerView *pickerView = [ [ UIPickerView alloc ]
initWithFrame: CGRectMake(0, 280, 320, 200)];
[ pickerView setDelegate: self ];
```

Сбор свойств выборщика

Чтобы переключить выводимые звуки щелчков, как правило, слышимых при прокручивании выборщика, воспользуйтесь методом setSoundsEnabled:

```
[ pickerView setSoundsEnabled: YES ];
```

Если выборщик должен позволять выбор нескольких элементов, воспользуйтесь методом setAllowsMultipleSelection:

```
[ pickerView setAllowsMultipleSelection: YES ];
```

Таблица выборщика

После создания вида выборщика вы должны указать ему создать лежащий в основе объект UIPickerViewTable.

Этот объект хранит различные элементы и структуру таблицы, используемую выбором:

```
UIPickerTable *pickerTable = [ pickerView
    _createTableWithFrame:
    CGRectMake(0, 0, 320, 200)
    forComponent: nil
];
```

Как и в случае с `UITableView`, таблица выбора должна иметь хотя бы один столбец. Он создается как объект `UITableColumn` и назначается виду выбора с помощью метода `columnForTable:`:

```
column = [ [ UITableColumn alloc ]
    initWithTitle: @"Column title"
    identifier:@"mycolumn"
    width: 320
];

[ pickerView columnForTable: column ];
```

После создания вида, таблицы и столбца в классе источника данных необходимо создать три метода:

- ❑ `numberOfColumnsInPickerView` — этот метод должен возвращать в качестве значения 1, если вы создаете многостолбцовый вид выбора;
- ❑ `numberOfRowsInColumn` — в виде выбора каждый столбец может иметь различное количество строк. Этот метод должен возвращать полное количество строк для столбца с указанным номером;
- ❑ `tableViewCellForRowAtIndexPath` — этот метод возвращает действительное количество объектов `UITableViewCell` для каждой строки в таблице. Как правило, это просто пустые объекты ячеек с заголовком. Однако можно использовать и более сложные классы ячеек таблицы, например, класс `UIImageAndTextTableViewCell`, рассмотренный в главе 3.

Прототипы и функция для этих методов будут показаны в примере далее в этой главе.

Отображение выборщика

Создав и сконфигурировав вид выборщика, а также написав привязку данных, вы подготовились к прикреплению выборщика к вашему управляющему виду:

```
[ mainScreen addSubview: pickerView ];
```

Считывание выборщика

Чтобы получить порядковый номер выбранного столбца в виде выборщика, воспользуйтесь методом `selectedRowForColumn` вида:

```
int selectedRow = [ pickerView selectedRowForColumn: 0 ];
```

Поскольку таблица выборщика создается самим видом выборщика, то трудно наследовать таблицу для использования метода `tableViewSelected`. Более удобный способ считывать значение в режиме реального времени — это создать подкласс класса `UIPickerView`, а затем подменить метод `mouseDown`:

```
- (void)mouseDown:(struct __GSEvent *)event {
    int selectedRow = [ self selectedRowForColumn: 0 ];
    [ super mouseDown: event ];
}
```

Пример: выбор типа вашего носа

В этом примере создается и представляется пользователю список различных типов носа. Сначала создается управляющий вид, который затем принимает в качестве подвида вид выборщика. Вы сможете прокручивать список носов и выбрать из них один.

Чтобы скомпилировать данный пример из командной строки, воспользуйтесь пакетом инструментов следующим образом:

```
$ arm-apple-darwin9-gcc -o MyExample MyExample.m -lobjc \
    -framework Foundation -framework CoreFoundation -framework UIKit
```

Листинги 7.19 и 7.20 содержат код.

Листинг 7.19. Пример выборщика (MyExample.h)

```
#import <CoreFoundation/CoreFoundation.h>
#import <Foundation/Foundation.h>
#import <UIKit/UIKit.h>
#import <UIKit/UIPickerView.h>
#import <UIKit/UIPickerTable.h>
#import <UIKit/UITableColumn.h>

@interface MainView : UIView
{
    UIPickerView *pickerView;
    UIPickerViewTable *pickerTable;
    NSMutableArray *cells;
    UITableColumn *column;
}

- (id)initWithFrame:(CGRect)rect;
- (void)dealloc;
@end

@interface MyApp : UIApplication
{
    UIWindow *window;
    MainView *mainView;
}

- (void)applicationDidFinishLaunching:(NSNotification *)aNotification;
@end
```

Листинг 7.20. Пример выборщика (MyExample.m)

```
#import <UIKit/UIPickerTableCell.h>
#import "MyExample.h"

int main(int argc, char **argv)
{
```

```
return UIApplicationMain(argc, argv, @"MyApp", @"MyApp");
}

@implementation MyApp

- (void)applicationDidFinishLaunching:(NSNotification *)aNotification {
    window = [ [ UIWindow alloc ] initWithContentRect:
        [ UIHardware fullScreenApplicationContentRect ]
    ];

    CGRect rect = [ UIHardware fullScreenApplicationContentRect ];
    rect.origin.x = rect.origin.y = 0.0f;

    mainView = [ [ MainView alloc ] initWithFrame: rect ];

    [ window setContentView: mainView ];
    [ window orderFront: self ];
    [ window makeKey: self ];
    [ window _setHidden: NO ];
}

@end

@implementation MainView

- (id)initWithFrame:(CGRect)rect {
    self = [ super initWithFrame: rect ];
    if (nil != self) {
        UIPickerViewTableCell *cell;
        NSMutableArray *noses = [ [ NSMutableArray alloc ] init ];
        int i;

        /* Создаем несколько носов */
        [ noses addObject: @"Straight" ];
        [ noses addObject: @"Aquiline" ];
        [ noses addObject: @"Retrousse" ];
        [ noses addObject: @"Busque" ];
    }
}
```

```

[ noses addObject: @"Sinuous" ];
[ noses addObject: @"Melanesian" ];
[ noses addObject: @"African" ];

cells = [ [ NSMutableArray alloc ] init ];
for(i=0;i<[ noses count ];i++) {
    cell = [ [ UIPickerViewTableCell alloc ]
        initWithFrame: CGRectMake(0, 0, 320, 80) ];
    [ cell setTitle: [ noses objectAtIndex: i ] ];
    [ cells addObject: cell ];
}

pickerView = [ [ UIPickerView alloc ]
    initWithFrame:
        CGRectMake(0, 280, 320, 200)];
[ pickerView setDelegate: self ];
[ pickerView setSoundsEnabled: NO ];

pickerTable = [ pickerView _createTableWithFrame:
    CGRectMake(0, 0, 320, 200) forComponent: nil ];

column = [ [ UITableColumn alloc ]
    initWithTitle: @"Nose"
    identifier:@"nose"
    width: rect.size.width
];

[ pickerView columnForTable: column ];
[ self addSubview: pickerView ];
}

return self;
}
- (int) pickerView:(id)pickerView numberOfRowsInColumn:(int)col
}
return !;

```

```
return 1;
}

- (int) pickerView:(id)pickerView numberOfRowsInColumn:(int)col
{
    return [ cells count ];
}

- (id) pickerView:(id)pickerView tableViewForRow:(int)row
    inColumn:(int)col
{
    return [ cells objectAtIndex: row ];
}

- (void)dealloc
{
    [ self dealloc ];
    [ super dealloc ];
}

@end
```

Как это работает

1. При порождении приложения создается новый управляющий вид `MainView` и вызывается его метод `initWithFrame`.
2. Этот метод создает массив видов носов и объект `UITableViewCell` для каждого из них, проталкивая ячейку в массив, используемый в привязке данных.
3. Создается `UIPickerView`, а его метод `_createTableWithFrame` используется для создания `UIPickerTable`, чтобы хранить содержимое таблицы. Затем к выборщику добавляется один столбец. А потом вид прикрепляется к основному виду.
4. После своей визуализации выборщик запрашивает источник данных на получение количества столбцов и строк, а затем указывает ячейкам таблицы отобразиться.

Для дальнейшего изучения

Проверьте наличие прототипов `UIPickerView.h` и `UIPickerTable.h` в папке `include` пакета инструментов. Их можно найти в папке `/toolchain/sys/usr/include/UIKit`.

Выборщик даты и времени

Класс `UIDatePicker` является подклассом `UIPickerView`. Он позволяет выбирать даты, время и длительность из настраиваемого независимого интерфейса выборщика. Выборщик даты автоматически настраивает его столбцы в соответствии с указанным стилем, поэтому при создании новых экземпляров не нужно выполнять никакой работы на низком уровне. Кроме того, он может быть настроен на любой диапазон дат и с любыми начальной и конечной датами.

`UIDatePicker` в большой степени полагается на класс `NSDate`, который является частью основополагающего набора классов, используемого в Cocoa на настольных системах. Более подробную информацию об этом классе можно найти в справочной информации о Cocoa от Apple на Web-узле [Apple Developer Connection](#). В используемом здесь примере мы создадим `NSDate` с помощью его простейшего метода — `initWithString:`

```
NSDate * myDate = [ [ NSDate alloc ]
    initWithString: @"1963-11-22 12:30:00 -0500" ];
```

Создание выборщика даты и времени

`UIDatePicker` — гораздо более простой, нежели стандартный `UIPickerView`. Он строит собственный источник данных на основе указанного вами диапазона дат. Чтобы воспользоваться им, просто создайте объект:

```
UIDatePicker *datePicker = [ [ UIDatePicker alloc ]
    initWithFrame: CGRectMake(0, 280, 320, 200)];
```

По умолчанию выборщик представляет текущую дату и время и позволяет пользователю выбирать любой месяц и время. Дальнейшие настройки операций выборщика рассмотрены в следующих подразделах.

Режим DatePicker

Выборщик дат поддерживает четыре различных режима выбора. Режим задается с помощью метода `setDatePickerMode:`

```
[ datePicker setDatePickerMode: 2 ];
```

Поддерживаются режимы, перечисленные в табл. 7.3.

Таблица 7.3

Параметр	Описание
0	Выбор часа, минуты и А.М. или Р.М.
1	Выбор месяца, дня и года
2	Выбор недели, месяца, дня, времени и А.М. или Р.М.
3	Выбор обычной временной длительности; количество часов и количество минут

Выделение "Today"

Чтобы выделить (подсветить) текущий день в выборщике, воспользуйтесь методом `setHighlightsToday`. Это приведет к тому, что для текущего дня будет отображено и подсвечено синим цветом слово "Today":

```
[ datePicker setHighlightsToday: YES ];
```

Временные интервалы

Минутный циферблат может быть настроен на отображение минут в одноминутных или пятиминутных интервалах, по умолчанию — одноминутных. Чтобы выбрать пятиминутные интервалы, используется метод `setStaggerTimeIntervals:`

```
[ datePicker setStaggerTimeIntervals: YES ];
```

Диапазоны дат

Диапазон разрешенных дат можно задать с помощью методов `setMinimumDate` и `setMaximumDate`. Если пользователь попытается прокру-

тять до какой-либо даты, лежащей вне этого интервала, то циферблат прокрутится обратно до ближайшей корректной даты. Оба метода принимают объект `NSDate`:

```
NSDate *minDate = [ [ NSDate alloc ]
    initWithString: @"1773-12-16 12:00:00 -0500" ];
NSDate *maxDate = [ [ NSDate alloc ]
    initWithString: @"1776-07-04 12:00:00 -0500" ];

[ datePicker setMinimumDate: minDate ];
[ datePicker setMaximumDate: maxDate ];
```

Если не заданы один или оба из этих методов, то по умолчанию пользователю разрешается выбирать любую дату в прошлом или будущем.

Чтобы задать дату, которую вы хотите отображать по умолчанию, воспользуйтесь методом `setDate`:

```
[ datePicker setDate: maxDate ];
```

Отображение выборщика даты

После создания выборщика его можно подключить к объекту вида точно так же, как и `UIPickerView`:

```
[ mainWindow addSubview: datePicker ];
```

По умолчанию высота выборщика составляет 200 пикселей, независимо от передаваемого в него размера фрейма. Вам необходимо убедиться в том, что для его размещения вы выделили достаточное пространство на экране.

Считывание даты

Дата, как правило, считывается из выборщика даты, когда пользователь переходит к другому виду, например, при покидании таблицы предпочтений. Хотя этот класс можно наследовать тем же способом, что и `UIPickerView` (чтобы подменить его события `mouseDown`), однако большинство приложений будут считывать дату после того, как пользователь нажмет кнопку перехода назад или какую-либо другую кнопку навигации.

Класс `UIDatePicker` из своего метода даты возвращает объект `NSDate`:

```
NSDate *selectedDate = [ datePicker date ];
```

Пример: независимый выборщик даты

Данный пример иллюстрирует использование основного объекта выбора даты для осуществления выбора даты в диапазоне между датой Бостонского чаепития (Boston Tea Party, 16 декабря 1773 г.) и Днем независимости Америки (American Independence Day, 4 июля 1776 г.). Пример просто создает объект `UIDatePicker` и отображает его пользователю:

Чтобы скомпилировать данный пример из командной строки, воспользуйтесь пакетом инструментов следующим образом:

```
$ arm-apple-darwin9-gcc -o MyExample MyExample.m -lobjc \
  -framework Foundation -framework CoreFoundation -framework UIKit
```

Листинги 7.21 и 7.22 содержат соответствующий код.

Листинг 7.21. Пример выбора даты и времени (MyExample.h)

```
#import <CoreFoundation/CoreFoundation.h>
#import <Foundation/Foundation.h>
#import <UIKit/UIKit.h>
#import <UIKit/UIDatePicker.h>

@interface MainView : UIView
{
    UIDatePicker *datePicker;
}

- (id)initWithFrame:(CGRect)rect;
- (void)dealloc;

@end

@interface MyApp : UIApplication
{
    UIWindow *window;
    MainView *mainView;
}

- (void)applicationDidFinishLaunching:(NSNotification *)aNotification;
@end
```

Листинг 7.22. Пример выбора даты и времени (MyExample.m)

```
#import <Foundation/Foundation.h>
#import <CoreFoundation/CoreFoundation.h>
#import <UIKit/UIPickerTableCell.h>
#import "MyExample.h"

int main(int argc, char **argv)
{
    return UIApplicationMain(argc, argv, @"MyApp", @"MyApp");
}

@implementation MyApp

- (void)applicationDidFinishLaunching:(NSNotification *)aNotification {
    window = [ [ UIWindow alloc ] initWithContentRect:
        [ UIHardware fullScreenApplicationContentRect ]
    ];

    CGRect rect = [ UIHardware fullScreenApplicationContentRect ];
    rect.origin.x = rect.origin.y = 0.0f;

    mainView = [ [ MainView alloc ] initWithFrame: rect ];

    [ window setContentView: mainView ];
    [ window orderFront: self ];
    [ window makeKey: self ];
    [ window _setHidden: NO ];
}

@end

@implementation MainView

- (id)initWithFrame:(CGRect)rect {
    self = [ super initWithFrame: rect ];
    if (nil != self) {
        datePicker = [ [ UIDatePicker alloc ]
```

```
initWithFrame:
    CGRectMake(0, 280, 320, 200)];

NSDate *minDate = [ [ NSDate alloc ]
    initWithString: @"1773-12-16 12:00:00 -0500" ];
NSDate *maxDate = [ [ NSDate alloc ]
    initWithString: @"1776-07-04 12:00:00 -0500" ];

[ datePicker setMinimumDate: minDate ];
[ datePicker setMaximumDate: maxDate ];
[ datePicker setDatePickerMode: 1 ];
[ datePicker setStaggerTimeIntervals: YES ];
[ datePicker setDelegate: self ];
[ datePicker setDate: maxDate ];
[ self addSubview: datePicker ];
}

return self;
}

- (void)dealloc
{
    [ self dealloc ];
    [ super dealloc ];
}

@end
```

Как это работает

1. При порождении приложения создается основной управляющий вид под названием `MainView` и вызывается его метод `initWithFrame`.
2. Создается объект `UIDatePicker` и назначаются минимальная и максимальная даты. Также задаются различные параметры для настройки отображения.
3. Выборщик даты добавляется к основному виду, в котором он отображается пользователю.

Для дальнейшего изучения

Проверьте наличие прототипов UIDatePicker.h в каталоге include пакета инструментов. Их можно найти в папке /toolchain/sys/include/UIKit.

Панели инструментов

Панели инструментов (toolbars), называемые также панелями кнопок, являются одним из решений Apple для универсальных устройств, не имеющих настоящих кнопок. Многие из функционально богатых приложений iPhone имеют четыре или пять важных функций, доступ к которым пользователю необходимо получать быстро. Расположенные внизу экрана, панели инструментов предоставляют то, что традиционно рассматривается как ярлыки. Возвращаясь к метафоре Apple с книгой, панели инструментов являются закладками к различным главам книги.

Многие из поставляемых приложений iPhone, включая приложения Mobile Phone, YouTube и iTunes WiFi Music Store, используют панели инструментов. Они служат для разделения соответствующих страниц с данными (например, Featured Music, Purchased Songs и пр.) и для предоставления ярлыков к различным функциям в пределах единого приложения (например, **Contacts** (Контакты), **Recent Calls** (Вызовы), **Voicemail** (Голосовая почта) и пр.)

Создание панели инструментов

Панели инструментов в UIKit представлены классом UIToolbar, хотя и была добавлена обратная совместимость, поэтому вы также можете использовать UIButtonBar, который является тем, что в предыдущих версиях iPhone называлось встроенным программным обеспечением. Как и панели навигации, панели инструментов в своем представлении являются относительно автономными. Внутри они обрабатывают выборы всех этих кнопок — и все это работает:

```
UIToolbar *toolBar = [ [ UIToolbar alloc ]
    initWithFrame: CGRectMake(0.0, 411.0, 320.0, 49.0)
    withItemList: [ self toolBarItemList ] ];
```

```
[ toolbar setDelegate: self ];  
[ toolbar setBarStyle: 1 ];  
[ toolbar setBarButtonTrackingMode: 2 ];
```

Приведенный фрагмент кода создает объект `UIToolbar` и назначает ему область отображения, расположенную вдоль нижней части окна. Самой панели инструментов необходим список элементов, который является массивом отображаемых на панели кнопок. Вместо того чтобы предоставлять этот массив вместе с кодом, используется метод, возвращающий такой массив. Создайте метод `toolbarItemList`, определяющий все кнопки панели инструментов и их свойства:

```
- (NSArray *)toolbarItemList {  
    return [ NSArray arrayWithObjects:  
        [ NSDictionary dictionaryWithObjectsAndKeys:  
            @"toolbarClicked:", kUIButtonBarButtonAction,  
            @"History.png", kUIButtonBarButtonInfo,  
            @"HistorySelected.png", kUIButtonBarButtonSelectedInfo,  
            [ NSNumber numberWithInt: 1], kUIButtonBarButtonTag,  
            self, kUIButtonBarButtonTarget,  
            @"Page 1", kUIButtonBarButtonTitle,  
            @"0", kUIButtonBarButtonType,  
            nil  
        ],  
        nil ];  
}
```

Данный метод создает массив словарных классов, чтобы содержать свойства для каждой кнопки. Этот массив ограничивается `nil` — пустым элементом. Каждый словарный объект содержит приведенную далее информацию о кнопке. Ваш файл заголовков должен объявлять переменные как `extern`, поскольку они скрыты в окружении и не объявлены ни в одном из прототипов.

- `kUIButtonBarButtonAction` — метод, который вызывается при нажатии на данную кнопку. Все кнопки могут вызывать одну и ту же процедуру `toolbarClicked`, поскольку кнопки могут иметь уникальные идентификаторы. Это будет продемонстрировано в приводимом далее примере.
- `kUIButtonBarButtonInfo` — имя файла изображения, который ассоциирован с кнопкой в ее обычном (ненажатом) состоянии. Это изображение должно быть скопировано в программную папку приложения, как обсуж-

далось в *главе 2*. Данный пример задает файл `History.png` — изображение, часто используемое в поставляемых Apple приложениях.

- ❑ `kUIButtonBarButtonSelectedInfo` — имя файла изображения, который используется, когда кнопка находится в нажатом состоянии. Он также должен присутствовать в программной папке приложения.
- ❑ `kUIButtonBarButtonTag` — тег (`tag`), специальный объект, который может передаваться для идентификации кнопки. В данном примере тег рассматривается как номер кнопки и имеет тип целого числа. Когда обрабатывается нажатие кнопки, этот тег может использоваться для определения того, какая именно кнопка была нажата.
- ❑ `kUIButtonBarButtonTarget` — объект, который должен получить уведомление (задается `kUIButtonBarButtonAction`). В данном примере для разрешения вызывающему виду получать уведомление о нажатиях кнопок используется `self`.
- ❑ `kUIButtonBarButtonTitle` — текст заголовка, отображаемый под изображением кнопки.
- ❑ `kUIButtonBarButtonType` — тип создаваемой кнопки. Единственным корректным значением для кнопок панели инструментов является `0`.

Для каждой кнопки должен создаваться объект `NSDictionary`, заканчивающийся ограничивающим `nil`. Каждая определяемая кнопка должна быть включена в группу кнопок и получить какую-либо конфигурацию. Данный пример отображает пять кнопок. Экран iPhone имеет ширину в 320 пикселей, поэтому каждая кнопка должна быть шириной в 64 пикселя ($5 \times 64 = 320$). Для определения конфигурации группы кнопок можно использовать цикл:

```
int buttons[5] = { 1, 2, 3, 4, 5 };
int tag;
[ toolbar registerButtonGroup:0 withButtons:buttons withCount:5 ];
[ toolbar showButtonGroup: 0 withDuration: 0.0 ];
for(tag = 1; tag < 5; tag++) {
    [ [ toolbar viewWithTag: tag ]
      setFrame: CGRectMake((tag - 1) * 64.0, 1.0, 64.0, 48.0)
    ];
}
```

И последнее, задайте кнопку по умолчанию, чтобы отразить отображаемый текущий вид:

```
[ toolbar showSelectionForButton: 1 ];
```

Отображение панели инструментов

После того как вы создали панель инструментов, можете добавить ее к какому-либо виду. Этот вид может также при нажатии кнопки обрабатывать переходы к страницам новых видов. Как правило, используется основной вид, но если программа слишком сложна, то различные ее части могут использовать разные панели инструментов:

```
[ self addSubview: toolBar ];
```

Бэйджи панели инструментов

В некоторых случаях ваше приложение может захотеть информировать пользователя о новых элементах на какой-либо странице панели инструментов. Чтобы привлечь внимание пользователя, можно добавить к кнопке на панели инструментов бэйдж (badge), содержащий число или какой-либо другой текст. Приведенный далее пример отображает бэйдж с числом 3 на второй кнопке:

```
[ toolBar setBadgeValue:@"3" forButton: 2 ];
```

Перехват нажатий кнопок

Когда создается панель инструментов, каждая кнопка описывается словарным объектом. Свойство `kUIButtonBarButtonAction` задает метод, который необходимо вызвать при нажатии кнопки. Этот метод может быть написан для обслуживания всех кнопок, или же для каждой из них можно определить по отдельному методу. Данный метод написан в целевом объекте панели инструментов, как описано в `kUIButtonBarButtonTarget`:

```
- (void)toolBarClicked:(id) sender {  
    int button = [ sender tag ];  
  
    /* Здесь делайте что-нибудь */  
}
```

Пример: еще один подход к книге с текстом

В разд. "Пример: переворачивание страниц" главы 3 создавалось 10 страниц текста, а для их переворачивания применялась панель навигации. Данный

пример аналогичен, за исключением того, что мы будем использовать пять страниц, представляющих пять различных видов в приложении. Каждая страница будет управляться кнопкой на панели инструментов, которая при нажатии переворачивает страницы до соответствующей. Для выполнения такого перехода используется `UITransitionView`.

Чтобы запустить это приложение на вашем iPhone, вам потребуется предоставить два изображения `Button.png` и `ButtonSelected.png`, которые копируются в программную папку приложения. Иначе кнопки будут отображаться только с текстом. Для данного примера вы можете сами нарисовать эти графические изображения или же позаимствовать какие-либо из уже имеющихся в iPhone кнопок, например, `/Applications/YouTube.app/History.png` и `/Applications/YouTube.app/HistorySelected.png`.

Чтобы скомпилировать эту программу из командной строки, воспользуйтесь пакетом инструментов следующим образом:

```
$ arm-apple-darwin9-gcc -o MyExample MyExample.m -lobjc \
    -framework CoreFoundation -framework UIKit -framework Foundation
```

Листинги 7.23 и 7.24 содержат соответствующий код.

Листинг 7.23. Пример панели инструментов (MyExample.h)

```
#import <CoreFoundation/CoreFoundation.h>
#import <UIKit/UIKit.h>
#import <UIKit/UITransitionView.h>
#import <UIKit/UITextView.h>
#import <UIKit/UIToolbar.h>

#define MAX_PAGES 5

extern NSString *kUIButtonBarButtonAction;
extern NSString *kUIButtonBarButtonInfo;
extern NSString *kUIButtonBarButtonInfoOffset;
extern NSString *kUIButtonBarButtonSelectedInfo;
extern NSString *kUIButtonBarButtonStyle;
extern NSString *kUIButtonBarButtonTag;
extern NSString *kUIButtonBarButtonTarget;
extern NSString *kUIButtonBarButtonTitle;
```

```
extern NSString *kUIButtonBarButtonTitleVerticalHeight;
extern NSString *kUIButtonBarButtonTitleWidth;
extern NSString *kUIButtonBarButtonType;

@interface MainView : UIView
{
    UITransitionView *transView; /* Our transition */
    UIToolbar *toolBar; /* Our toolbar */

    /* Несколько страниц для перелистывания */
    UITextView *textPage[MAX_PAGES];
}

- (id)initWithFrame:(CGRect)frame;
- (void)dealloc;
- (void)flipTo:(int)page;
- (UIToolbar *)createBarButton;
- (void)toolBarClicked:(id)sender;
- (NSArray *)toolBarItemList;
@end

@interface MyApp : UIApplication
{
    UIWindow *window;
    MainView *mainView;
}

- (void)applicationDidFinishLaunching:(NSNotification *)aNotification;
@end
```

Листинг 7.24. Пример панели инструментов (MyExample.m)

```
#import "MyExample.h"

int main(int argc, char **argv)
{
```

```
NSAutoreleasePool *autoreleasePool = [
    [ NSAutoreleasePool alloc ] init
];
int returnCode = UIApplicationMain(argc, argv, @"MyApp", @"MyApp");
[ autoreleasePool release ];
return returnCode;
}

@implementation MyApp

- (void)applicationDidFinishLaunching:(NSNotification *)aNotification {
    window = [ [ UIWindow alloc ] initWithContentRect:
        [ UIHardware fullScreenApplicationContentRect ]
    ];

    CGRect rect = [ UIHardware fullScreenApplicationContentRect ];
    rect.origin.x = rect.origin.y = 0.0f;

    mainView = [ [ MainView alloc ] initWithFrame: rect ];

    [ window setContentView: mainView ];
    [ window orderFront: self ];
    [ window makeKey: self ];
    [ window _setHidden: NO ];
}
@end

@implementation MainView

- (id)initWithFrame:(CGRect)rect {
    self = [ super initWithFrame: rect ];
    if (nil != self) {
        CGRect viewRect;
        int i;

        viewRect = CGRectMake(rect.origin.x, rect.origin.y,
            rect.size.width, rect.size.height - 48.0);
    }
}
```

```
/* Создаем несколько объектов UITextView как различные виды */
for(i=0;i<MAX_PAGES;i++) {
    textPage[i] = [ [ UITextView alloc ],initWithFrame: viewRect ];
    [ textPage[i] setText: [ [ NSString alloc ] initWithFormat:
        @"Some text for page %d", i+1 ] ];
}

/* Создаем нашу UIToolbar */
toolBar = [ self createButtonBar ];
[ self addSubview: toolBar ];

/* Создаем переход, чтобы мы могли легко переключаться
   между страницами */
transView = [ [ UITransitionView alloc ] initWithFrame: viewRect ];
[ self addSubview: transView ];

/* Переход к первой странице */
[ self flipTo: 1 ];
}

return self;
}

- (void)dealloc
{
    [ self dealloc ];
    [ super dealloc ];
}

- (void)flipTo:(int)page {
    [ transView transition: 0 toView: textPage[page-1] ];
}

- (UIToolbar *)createButtonBar {
    UIToolbar *myButtonBar;
```

```
myButtonBar = [ [ UIToolbar alloc ]
    initWithFrame: CGRectMake(0.0f, 411.0f, 320.0f, 49.0f)
    withItemList: [ self toolBarItemList ] ];
[ myButtonBar setDelegate: self ];
[ myButtonBar setBarStyle: 1 ];
[ myButtonBar setButtonBarTrackingMode: 2 ];

int buttons[5] = { 1, 2, 3, 4, 5 };
[ myButtonBar registerButtonGroup: 0 withButtons: buttons
    withCount: 5 ];
[ myButtonBar showButtonGroup: 0 withDuration: 0.0 ];
int tag;

for(tag = 1; tag < 5; tag++) {
    [ [ myButtonBar viewWithTag: tag ]
        setFrame:CGRectMake(2.0f + ((tag - 1) * 63.0), 1.0, 64.0, 48.0f)
    ];
}
[ myButtonBar showSelectionForButton: 3 ];

return myButtonBar;
}

- (NSArray *)toolBarItemList {
return [ NSArray arrayWithObjects:
    [ NSDictionary dictionaryWithObjectsAndKeys:
        @"toolBarClicked:", kUIButtonBarButtonAction,
        @"Button.png", kUIButtonBarButtonInfo,
        @"ButtonSelected.png", kUIButtonBarButtonSelectedInfo,
        [ NSNumber numberWithInt: 1], kUIButtonBarButtonTag,
        self, kUIButtonBarButtonTarget,
        @"Page 1", kUIButtonBarButtonTitle,
        @"0", kUIButtonBarButtonType,
        nil
    ],
];
```

```
[ NSDictionary dictionaryWithObjectsAndKeys:
    @"toolBarClicked:", kUIButtonBarButtonAction,
    @"Button.png", kUIButtonBarButtonInfo,
    @"ButtonSelected.png", kUIButtonBarButtonSelectedInfo,
    [ NSNumber numberWithInt: 2], kUIButtonBarButtonTag,
    self, kUIButtonBarButtonTarget,
    @"Page 2", kUIButtonBarButtonTitle,
    @"0", kUIButtonBarButtonType,
    nil
],
```

```
[ NSDictionary dictionaryWithObjectsAndKeys:
    @"toolBarClicked:", kUIButtonBarButtonAction,
    @"Button.png", kUIButtonBarButtonInfo,
    @"ButtonSelected.png", kUIButtonBarButtonSelectedInfo,
    [ NSNumber numberWithInt: 3], kUIButtonBarButtonTag,
    self, kUIButtonBarButtonTarget,
    @"Page 3", kUIButtonBarButtonTitle,
    @"0", kUIButtonBarButtonType,
    nil
],
```

```
[ NSDictionary dictionaryWithObjectsAndKeys:
    @"toolBarClicked:", kUIButtonBarButtonAction,
    @"Button.png", kUIButtonBarButtonInfo,
    @"ButtonSelected.png", kUIButtonBarButtonSelectedInfo,
    [ NSNumber numberWithInt: 4], kUIButtonBarButtonTag,
    self, kUIButtonBarButtonTarget,
    @"Page 4", kUIButtonBarButtonTitle,
    @"0", kUIButtonBarButtonType,
    nil
],
```

```
[ NSDictionary dictionaryWithObjectsAndKeys:
    @"toolBarClicked:", kUIButtonBarButtonAction,
```

```
        @"Button.png", kUIButtonBarButtonInfo,  
        @"ButtonSelected.png", kUIButtonBarButtonSelectedInfo,  
        [ NSNumber numberWithInt: 5], kUIButtonBarButtonTag,  
        self, kUIButtonBarButtonTarget,  
        @"Page 5", kUIButtonBarButtonTitle,  
        @"0", kUIButtonBarButtonType,  
        nil  
    ],  
  
    nil  
];  
}  
  
-(void)toolBarClicked:(id) sender {  
    int button = [ sender tag ];  
    [ self flipTo: button ];  
}  
@end
```

Как это работает

Панели инструментов — достаточно сложные штуковины. Вот как они работают:

1. При порождении приложения создается объект `MainView` и вызывается его метод `initWithFrame`. Для создания небольшой области `viewRect`, учитывающей высоту панели инструментов, используется область отображения. Она служит для создания текстового и переходного видов для верхней части экрана, расположенной над панелью кнопок.
2. Метод `initWithFrame` вызывает метод `createButtonBar`. Он создает объект `UIToolbar` и назначает ему массив кнопок посредством метода `toolbarItemList`. Этот массив содержит словарь для каждой отображаемой кнопки, определяющий ее заголовки, теги, изображения и методы, которые должны вызываться при ее нажатии. Поскольку наши кнопки выполняют сопоставимые действия (отличаются только отображаемые страницы), то все они вызывают один и тот же метод `toolBarClicked`.

3. Конфигурация каждой кнопки определяется в группе кнопок и регистрируется с панелью инструментов. Затем панели инструментов указывается отобразить эту группу.
4. Панель инструментов добавляется к основному виду, где она и отображается.
5. Создаются пять объектов `UITextView`. Они играют роли примеров пяти кнопочных видов. `initWithFrame` вызывает метод `flipTo`, который отвечает за переворачивание страниц до переданного в него номера страницы.
6. При нажатии кнопки вызывается метод `toolbarClicked`. Он захватывает идентификационный тег кнопки и снова вызывает `flipTo` для перелистывания к новой странице. Обновление панели инструментов является функцией самой панели инструментов и выполняется автоматически.

Для дальнейшего изучения

- Поэкспериментируйте с размещением панели инструментов. Можно ли поместить ее наверху экрана? Могут ли в одном виде существовать несколько панелей инструментов?
- Панель инструментов может обрабатывать любое количество кнопок. Измените этот пример и используйте три большие кнопки, а затем попробуйте использовать восемь.
- Какого рода теги могут быть назначены кнопке? Попробуйте прикрепить различные объекты.
- Проверьте наличие прототипов `UIToolbar.h` в папке пакета инструментов. Они могут находиться в папке `include` вашего пакета инструментов: `/toolchain/sys/include/UIKit`.

Изменения ориентации

iPhone оснащен аппаратным обеспечением для определения его положения в окружающем мире. В частности, один из сенсоров, акселерометр, способен определять положение, в котором держат iPhone. Приложениям, которым необходимо предоставлять альбомный режим, очень важно знать, как считать ориентацию и что с ней делать при ее изменении.

Считывание ориентации

Ориентация (orientation) iPhone может быть считана с помощью статического метода `deviceOrientation`, находящегося в классе `UIHardware`:

```
int orientation = [ UIHardware deviceOrientation: YES ];
```

Этот метод возвращает одну из шести различных возможных ориентаций (табл. 7.4), указывая то, как держат iPhone в настоящее время.

Таблица 7.4

Ориентация	Описание
0	<code>kOrientationFlatUp</code> — устройство лежит на плоской поверхности лицевой стороной вверх
1	<code>kOrientationVertical</code> — устройство держат вертикально правой стороной вверх
2	<code>kOrientationVerticalUpsideDown</code> — устройство держат вертикально вверх дном
3	<code>kOrientationHorizontalLeft</code> — устройство наклонено влево
4	<code>kOrientationHorizontalRight</code> — устройство наклонено право
5	<code>kOrientationUnknown</code> — положение устройства неизвестно, сбой сенсора
6	<code>kOrientationFlatDown</code> — устройство лежит на плоской поверхности лицевой стороной вниз

Сенсор может быть считан в начале запуска приложения, но полезнее знать, когда ориентация поменяется. Изменение в ориентации сообщается автоматически в класс `UIApplication`, от которого порождено ваше приложение GUI. Для перехвата этого события может быть подменен метод `deviceOrientationChanged`:

```
- (void)deviceOrientationChanged:(GSEvent *)event {
    int newOrientation = [ UIHardware deviceOrientation: YES ];
    /* Ориентация изменилась, сделайте что-нибудь */
}
```

Например, если возвращенное значение соотносится с альбомным режимом (`kOrientationHorizontalLeft` или `kOrientationHorizontalRight`), то приложение может предпринять соответствующие шаги для переключения в альбомный режим. Один из способов сделать это — создать классы `UIView` для обслуживания книжного или альбомного видов по отдельности. Затем при изменении ориентации эти два вида могут переходить вперед и назад:

```
[ transitionView transition: 0
  fromView: portraitView toView: landscapeView
];
```

Вращающиеся объекты

Базовый класс `UIView` поддерживает метод `setRotationBy`, который позволяет практически любому отображаемому объекту в `UIKit` вращаться, чтобы приспособиваться к различным ориентациям:

```
[ textView setRotationBy: 90 ];
```

Передаваемый аргумент определяет угол в градусах, на который нужно повернуть объект.

Для того чтобы прийти в соответствие с ориентацией iPhone, вращаться нужно не только различным объектам, строка состояния также может вращаться. Воспользуйтесь методом `setStatusBarItemMode`, который обсуждался в *главе 3*:

```
[ self setStatusBarItemMode: 0 orientation: 90 duration: 0
  fenceID: nil animation: 0 ];
```

В зависимости от того, были ли объекты повернуты для приспособления к левому или к правому повороту iPhone, укажите либо 90° , либо -90° соответственно.

Размеры окна объектов будут изменены, чтобы соответствовать ориентации этих объектов, поэтому начальная точка объектов также будет смещена. Например, чтобы отобразить текстовый вид в альбомном режиме, воспользуйтесь фреймом, задаваемым в альбомном разрешении:

```
CGRect textRect = CGRectMake(-90, 70, 480, 300);
textView = [ [ UITextView alloc ] initWithFrame: textRect ];
[ textView setRotationBy: 90 ];
[ self addSubview: textView ];
```

Пример: поворот мира в другую сторону

В *главе 3* одним из самых первых приведенных нами примеров было приложение "Hello, World!" Мы воспользуемся этим простым примером для демонстрации простейшего альбомного поворота экрана. Приведенный далее код выводит на экран приложение "Hello, World!", используя строку состояния альбомного режима и поворачивая текстовое поле в соответствии с ним.

Чтобы скомпилировать этот пример из командной строки, воспользуйтесь пакетом инструментов следующим образом:

```
$ arm-apple-darwin9-gcc -o MyExample MyExample.m -lobjc \
  --framework CoreFoundation -framework UIKit
```

Листинги 7.25 и 7.26 содержат соответствующий код.

Листинги 7.25. Пример ориентации (MyExample.h)

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

@interface MainView : UIView
{
    UITextView *textView;
    CGRect rect;
}
- (id)initWithFrame:(CGRect)_rect;
- (void)dealloc;
@end

@interface MyApp : UIApplication
{
    UIWindow *window;
    MainView *mainView;
}
- (void)applicationDidFinishLaunching:(NSNotification *)aNotification;
@end
```

Листинги 7.26. Пример ориентации (MyExample.m)

```
#import "MyExample.h"

int main(int argc, char **argv)
{
    return UIApplicationMain(argc, argv, @"MyApp", @"MyApp");
}

@implementation MyApp

- (void)applicationDidFinishLaunching:(NSNotification *)aNotification {
    UIWindow *window = [ [ UIWindow alloc ] initWithContentRect:
        [ UIScreen fullScreenApplicationContentRect ]
    ];

    CGRect rect = [ UIScreen fullScreenApplicationContentRect ];
    rect.origin.x = rect.origin.y = 0.0f;

    UIView *mainView = [ [ MainView alloc ] initWithFrame: rect ];
    [ self setStatusBarMode: 0 orientation: 90 duration: 0 ];

    [ window setContentView: mainView ];
    [ window orderFront: self ];
    [ window makeKey: self ];
    [ window _setHidden: NO ];
}

@end

@implementation MainView

- (id)initWithFrame:(CGRect)_rect {

    self = [ super initWithFrame: _rect ];
    if (nil != self) {
        rect = _rect;
    }
}

@end
```

```
CGRect textRect = CGRectMake(-90, 70, 480, 300);
textView = [ [ UITextView alloc ] initWithFrame: textRect ];
[ textView setRotationBy: 90 ];
[ textView setText: @"Hello, World!" ];
[ self addSubview: textView ];
}
return self;
}

- (void)dealloc
{
    [ self dealloc ];
    [ super dealloc ];
}
@end
```

Как это работает

1. При порождении приложения создается объект `MainView` и вызывается его метод `initWithFrame`.
2. Основной вид создает класс `UITextView` с разрешением 480×300 для альбомного режима.
3. Вызывается метод `setRotationBy` текстового вида для поворота объекта на 90° по часовой стрелке. Затем он добавляется на экран.

Считывание акселерометра

API ориентации получает всю информацию от небольшого акселерометра, встроенного в iPhone. Эта крошечная часть аппаратного обеспечения сообщает координаты X , Y , Z положения устройства. API ориентации существенно упрощает такую выходную информацию и переводит ее в положения, удобные для использования и понимания, однако более смелые разработчики могут считывать необработанные данные напрямую.

Эрлинг Эллинген (Erling Ellingsen) потратил огромное количество времени на разборку процедур, которые взаимодействуют с акселерометром, и неожиданно обнаружил, что основной класс приложения, `UIApplication`, от-

правляет частые уведомления о состоянии акселерометра. Для перехвата этих уведомлений подмените метод `acceleratedInX:`:

```
- (void)acceleratedInX:(float)xAxis Y:(float)yAxis Z:(float)zAxis {  
    /* Акселерометр как X-Axis, Y-Axis и Z-Axis */  
}
```

Поскольку акселерометр iPhone не включает в себя гироскоп, то он не может предоставлять информацию о скорости или столько информации о состоянии устройства, сколько может предоставлять, к примеру, контроллер Nintendo Wii. Однако он доказал свою полезность для простых приложений, например, тех, которые используют распознавание дрожания iPhone.

Для дальнейшего изучения

Проверьте наличие прототипов `UIView-Geometry.h` в каталоге `include` пакета инструментов. Их можно найти в папке `/toolchain/sys/include/UIKit`.

Виды Web-документа и прокрутки

В главе 3 мы познакомились с объектом `UITextView` и его методом `setHTML` для создания окон с HTML-форматированием. Объект `UIWebView` создает вокруг `UITextView` мир, подобный обозревателю, и добавляет множество простейших процедур, которые имеются в обозревателе Интернета: удаленная выборка страниц, прямая и обратная навигация, а также изменение масштаба. Один из основных и лучших компонентов, который заставляет работать Safari, `UIWebView`, может использоваться и в ваших приложениях. Web-виды (web views) могут отображать не только HTML-страницы, они также могут отображать PDF-файлы (локальные и удаленные), графические изображения и любые другие типы файлов, поддерживаемые Safari.

Шон Хебер (Sean Heber of iApp-a-Day) написал функциональную обертку для класса `UIWebView`, названную `SimpleWebView`. Далее вы увидите, как работает его класс и те усовершенствования, которые мы внесли в него.

Создание Web-вида

Функциональный Web-вид состоит из трех компонентов:

- объект `UIWebView` выполняет все выборки, масштабирование и обработку ссылок для данного вида;

- объект `UIScrollView` требуется для прокрутки Web-вида, особенно при увеличении его масштаба;
- объект `NSURLRequest` предоставлен в качестве класса, указывающего на ресурс, который необходимо выбрать.

Класс Шона `SimpleWebView` инкапсулирует все эти объекты в управляющий класс на основе `UIView`:

```
@interface SimpleWebView : UIView {
    UIWebView *webView;
    UIScrollView *scroller;
    NSURLRequest *urlRequest;
}
-(id)initWithFrame:(CGRect)frame;
-(void)loadURL:(NSURL *)url;
-(void)dealloc;
```

`SimpleWebView` подменяет метод `initWithFrame` базового класса. Кроме того, он добавляет новый метод `loadURL`, который используется для загрузки Web-страницы или файлового ресурса. Когда инициализируется оберточный класс Шона, создаются объекты `UIWebView` и `UIScrollView`. Затем загружается Web-вид как уровень для класса прокрутки.

Как работают прокрутки

Рассматривайте прокрутки (scrollers) как дешифрующий элемент. Поместив этот маленький элемент из пластика на какую-либо часть секретной шифрованной книги, вы откроете небольшую часть соответствующей страницы. Остальная часть этой страницы остается на том же месте, однако вы не сможете ее прочесть, пока не поместите на нее дешифрующий элемент. Такой элемент представляет собой экран iPhone, являющийся окном и единственным содержимым, которое может видеть пользователь. Остальная часть Web-страницы скрыта из поля зрения за пределами экрана до тех пор, пока пользователь не переместит окно к той части, которую он хочет увидеть. Создание прокрутки подобно созданию не только дешифрующего элемента, но и пустых страниц в секретной шифрованной книге. Web-вид — это содержимое, которое приклеивается к страницам для создания окна с прокручиваемым содержимым:

```
scroller = [ [ UIScrollView alloc ] initWithFrame: frame ];
[ scroller setScrollingEnabled: YES ];
```

```
[ scroller setAdjustForContentSizeChange: YES ];  
[ .scroller setClipsSubviews: YES ];  
[ scroller setAllowsRubberBanding: YES ];  
[ scroller setDelegate: self ];  
[ self addSubview: scroller ];
```

Класс `UIScrollView` может настраиваться множеством способов. Наиболее распространенными являются следующие параметры:

- `setScrollingEnabled: (BOOL)` — включает панели прокрутки, разрешая прокрутке выполнять свою работу;
- `setAdjustForContentSizeChange: (BOOL)` — автоматически перепрограммирует собственную панель прокрутки всякий раз, когда меняются границы содержимого;
- `setClipsSubviews: (BOOL)` — указывает прокрутке не обрезать никакие данные, которые будут в ней содержаться;
- `setAllowsRubberBanding: (BOOL)` — позволяет пользователю при достижении границы прокручиваемой области перетаскивать несколько дальше верхней и нижней границ. Когда пользователь поднимает свой палец, эта область отскакивает обратно на свое место подобно резиновой ленте, подавая тем самым визуальный сигнал о том, что пользователь достиг начала или конца документа;
- `setAllowsFourWayRubberBanding: (BOOL)` — как и предыдущий параметр, данный метод — метод резиновой ленты. По умолчанию только верхняя и нижняя границы страницы имеют эффект подобной резиновой ленты. Чтобы разрешить прокрутке применять эффект резиновой ленты ко всем четырем сторонам, задайте этот метод в дополнение к предыдущему;
- `setBottomBufferHeight: (float)` — данный метод буферизует содержимое таким образом, чтобы какая-либо определенная его часть была скрыта от конца прокручиваемой области. Воспользуйтесь этим, если у вашего содержимого большой буфер, например, завышенный серый фрейм или другая граница, которую вы бы хотели обрезать с экрана;
- `setContentSize: (CGSize)` — данный метод может использоваться для определения размера страниц содержимого, которые будут сцеплены с прокруткой. В классе `SimpleWebView` этот метод вызывается всякий раз, когда происходит изменение размеров изображения.

После того как вы создадите прокрутку, создайте объект `UIWebView` и добавьте его как подвид прокрутки. Тем самым будет склеено содержимое Web-вида и прокрутка, что даст прокрутке управление видимой областью Web-страницы:

```
webView = [ [ UIWebView alloc ]
    initWithFrame: [ scroller bounds ] ];
[ webView setTilingEnabled: YES ];
[ webView setTileSize: frame.size ];
[ webView setAutoresizes: YES ];
[ webView setDelegate: self ];
[ webView setEnabledGestures: 0xFF ];
[ webView setSmoothsFonts: YES ];
[ scroller addSubview: webView ];
```

В Web-виде устанавливаются следующие параметры:

- `setTilingEnabled`, `setTileSize`. `UITiledView` — это вид особого рода, используемый Google Maps, Safari и другими специализированными приложениями. Мозаичный вид разработан для загрузки содержимого в сетку, позволяющую отображать содержимое, даже если его загрузка еще не завершилась. Поскольку мозаичное размещение требуется Web-видами, то класс `UIWebView` без него работает некорректно, и не будет отображать какое-либо содержимое;
- `setAutoresize` — указывает Web-виду автоматически изменять свой размер при загрузке или изменении масштаба страницы;
- `setEnabledGestures` — разрешив сжатие и растяжение, с Web-видом можно обращаться так же, как и со страницей Safari. Затем можно подменить различные методы `UIResponder`, как описано в главе 4, для получения уведомлений об этих жестах;
- `setSmoothFonts` — указывает Web-виду сглаживать шрифты загружаемого содержимого.

Итак, Web-вид создан и добавлен к прокрутке. Пришло время вызывать метод `loadURL` и загрузить какой-либо ресурс. URL-запрос делается для того, чтобы загрузить содержимое в Web-вид. Объект `NSURL` задает адрес для выполнения загрузки. Этот объект может создаваться с помощью метода `initWithString` объекта `NSURL`:

```
NSURL *url = [ [ NSURL alloc ]
    initWithString: @"http://www.oreilly.com"
];
```

Внутри класса `SimpleWebView` метод `loadURL` берет объект `NSURL` и создает `NSURLRequest`. Объект `NSURLRequest` похож на `NSURL`, однако инкапсулирует такую информацию, как коды состояния и отклика, которые необходимы для отслеживания момента загрузки Web-страницы. Данный запрос передается напрямую Web-виду:

```
NSURLRequest *urlRequest = [
    [ NSURLRequest requestWithURL: url ] retain
];
[ webView loadRequest: urlRequest ];
```

Настройка прокруток

Поскольку размер Web-страницы был неизвестен при создании прокручиваемого вида, вы должны подменить два метода в классе `UIScrollView` так, чтобы класс `SimpleWebView` получал уведомления, когда содержимое будет загружено и отображено. Этими методами являются `didDrawInRect` и `didSetFrame`. Всякий раз, когда обновляется прикрепленное к прокручиваемому классу содержимое, уведомляется метод `didDrawInRect`, разрешая ему пересчитать содержимое. Затем он настраивает его панели прокрутки в соответствии с размерами содержимого. В результате этого вызывается метод `didSetFrame`, который задает границы содержимого прокрутки.

```
-(void)view: (UIView*)v didSetFrame:(CGRect)f
{
    if (v == webView) {
        [ scroller setContentSize: f.size ];
    }
}

-(void)view:(id)v didDrawInRect:(CGRect)f duration:(float)d
{
    if (v == webView) {
        CGSize size = [ webView bounds ].size;
        if (size.height != lastSize.height || size.width != lastSize.width)
        {
            lastSize = size;
            [ scroller setContentSize: size ];
        }
    }
}
```

Для отслеживания последнего переданного размера документа в приведенном примере в классе `SimpleWebView` используется структура `CGSize` под названием `lastSize`. Всякий раз, когда пользователь изменяет масштаб документа или же щелкает по ссылке, меняется размер документа. При этом для перенастройки панелей прокрутки должен вызываться метод `setContentSize` прокрутки.

Автоматическое сглаживание при изменении размеров

Одно из усовершенствований, которое было сделано в этом классе, — это возможность автоматически сглаживать изображение при изменении его масштаба. Для этого вам необходимо наследовать класс `UIWebView` для перехвата уведомлений, отправляемых при жестах или двойных касаниях:

```
@interface MyWebView : UIWebView
{
}
- (void)gestureEnded:(struct __GSEvent *)event;
- (void)doubleTap:(struct __GSEvent *)event;
@end
```

Затем эти два метода уведомляют делегата класса, который является объектом `SimpleWebView`, чтобы он мог еще раз обновить панели прокрутки:

```
- (void) gestureEnded:(struct __GSEvent *)event
{
    [ super gestureEnded: event ];
    [ _delegate gestureEnded: event ];
}

- (void) doubleTap:(struct __GSEvent *)event
{
    [ super doubleTap: event ];
    [ _delegate doubleTap: event ];
}
```

При изменении масштаба страницы имеющаяся на ней графика становится расплывчатой, если ее не сгладить. Для этого объект `SimpleWebView` вызывает метод `redrawScaledDocument`. Этот метод принадлежит классу

`uiwebDocumentView` и сглаживает при необходимости графику и шрифты страницы:

```
- (void)gestureEnded:(struct_____GSEvent *)event {
    [ webView redrawScaledDocument ];
    [ webView setNeedsDisplay ];
    [ scroller setContentSize: [ webView bounds ].size ];
}
t
```

Когда страница перерисовывается при масштабировании, `gestureEnded` вызывает метод `setNeedsUpdate` Web-вида, который удостоверяет, что все изменения распространяются и вне экрана. Размер содержимого также должен быть пересчитан, поскольку масштаб содержимого был изменен, а также на прокрутку должен быть вызван `setContentSize` для обновления панелей прокрутки. Эту задачу должны выполнять обе ваши функции: `gestureEnded` и `doubleTap`.

Использование класса *SimpleWebView*

К счастью, класс `SimpleWebView` гораздо легче использовать, чем понять. Чтобы создать экземпляр класса `SimpleWebView`, содержащий все эти части, основной вид вызывает метод `initWithFrame` класса, с последующим вызовом `loadURL`:

```
NSURL *url = [ [ NSURL alloc ]
    initWithString: @"http://www.oreilly.com"
];

SimpleWebView *webView = [ [ SimpleWebView alloc ]
    initWithFrame: rect ];
[ webView loadURL: url ];
```

Чтобы загрузить локальный файл, например, PDF, используйте URI `file://`:

```
NSURL *url = [ [ NSURL alloc ]
    initWithString: @"file:///var/root/Media/PDFs/Resume.PDF" ];
```

После создания объекта `SimpleWebView` он может быть добавлен к основному виду как подвид или перемещен как его собственный вид:

```
[ self addSubview: webView ];
```

Пример: простой обозреватель Интернета

Это один из самых интересных примеров в данной главе. Этот облегченный вариант обозревателя Интернета использует класс `SimpleWebView` Шона Хебера с внесенными нами усовершенствованиями совместно с адресной строкой, созданной из объекта `UITextView`. Кроме того, для осуществления ввода используется собственный всплывающий объект `UIKeyboard`. Когда пользователь вводит URL в адресную строку и нажимает кнопку **Go**, Web-вид загружает заданную страницу.

Данный пример использует некоторые подмененные методы для объекта `UITextView`. Ранее в этой главе мы рассмотрели клавиатуры и обсудили значение подмены метода `contentMouseUpInView`. Это используется для включения или отключения переключения отображения клавиатуры при касании вида. В данном примере мы познакомимся с новой подменой под названием `shouldInsertText`. Этот метод вызывается всякий раз, когда пользователь нажимает какую-либо клавишу. Данный пример использует ее для проверки, не нажата ли клавиша `<Return>` (обозначенная на клавиатуре как **Go**), и в случае, если она нажата, то указывает управляющему виду загрузить новую страницу.

Данный пример работает с Web-объектами и локальными файлами. Чтобы получить доступ к локальным файлам, надо удалить префикс протокола `http://`, предшествующий пути к файлу. Как обычно, дайте для загрузки Web-страниц несколько секунд. Чтобы не тратить на этот пример сотни страниц в этой книге, многие из эстетически привлекательных возможностей обозревателя Интернета, например, индикаторы загрузки страниц и симпатичные панели инструментов, были опущены. Помните, это — всего лишь простейший обозреватель Интернета.

Чтобы скомпилировать это приложение из командной строки, воспользуйтесь пакетом инструментов следующим образом:

```
$ arm-apple-darwin9-gcc -o MyExample MyExample.m SimpleWebView.m \
  -lobjc -framework Foundation -framework CoreFoundation \
  -framework UIKit
```

Листинги 7.27 и 7.28 содержат код Web-вида и прокрутки, а листинги 7.29 и 7.30 — код основного приложения и основного вида.

Листинги 7.27. Пример Web-вида и прокрутки (SimpleWebView.h)

```
/*
 * By: Sean Heber <sean@spiffytech.com>, J. Zdziarski
 * iApp-a-Day - November, 2007
 * BSD License
 */
#import <UIKit/UIKit.h>
#import <UIKit/UIScrollView.h>
#import <UIKit/UIWebView.h>

@interface MyWebView : UIWebView
{
}

- (void)gestureEnded:(struct __GSEvent *)event;
- (void)doubleTap:(struct __GSEvent *)event;
@end

@interface SimpleWebView : UIView {
    MyWebView *webView;
    UIScrollView *scroller;
    NSURLRequest *urlRequest;
    CGSize lastSize, size;
}

- (id)initWithFrame: (CGRect)frame;
- (id)loadURL: (NSURL *)url;
- (void)dealloc;
@end
```

Листинг 7.28. Пример Web-вида и прокрутки (SimpleWebView.m)

```
/* By: Sean Heber <sean@spiffytech.com>, J. Zdziarski
 * iApp-a-Day - November, 2007
 * BSD License */
#import "SimpleWebView.h"
```

```
#import <UIKit/UIView-Geometry.h>
#import <UIKit/UIView-Rendering.h>

@implementation MyWebView
- (void) gestureEnded:(struct __GSEvent *)event
{
    [ super gestureEnded: event ];
    [ _delegate gestureEnded: event ];
}

- (void) doubleTap:(struct __GSEvent *)event
{
    [ super doubleTap: event ];
    [ _delegate doubleTap: event ];
}
@end

@implementation SimpleWebView

-(void)view: (UIView*)v didSetFrame:(CGRect)f
{
    if (v == webView) {
        [ scroller setContentSize: f.size ];
    }
}

-(void)view:(id)v didDrawInRect:(CGRect)f duration:(float)d
{
    if (v == webView) {
        size = [ webView bounds ].size;
        if (size.height != lastSize.height || size.width != lastSize.width)
        {
            lastSize = size;
            [ scroller setContentSize: size ];
        }
    }
}
}
```

```
- (void)gestureEnded:(struct __GSEvent *)event {
    [ webView redrawScaledDocument ];
    [ webView setNeedsDisplay ];
    [ scroller setContentSize: [ webView bounds ].size ];
}

- (void)doubleTap:(struct __GSEvent *)event {
    struct timeval tv;
    tv.tv_sec = 2;
    tv.tv_usec = 0;
    select(NULL, NULL, NULL, NULL, &tv);
    [ webView redrawScaledDocument ];
    [ webView setNeedsDisplay ];
    [ scroller setContentSize: [ webView bounds ].size ];
}

-(void)dealloc
{
    [ URLRequest release ];
    [ webView release ];
    [ scroller release ];
    [ super dealloc ];
}

-(id)initWithFrame: (CGRect)frame
{
    [ super initWithFrame: frame ];

    scroller = [ [ UIScrollView alloc ] initWithFrame: frame ];
    [ scroller setScrollingEnabled: YES ];
    [ scroller setAdjustForContentSizeChange: YES ];
    [ scroller setClipsSubviews: NO ];
    [ scroller setAllowsRubberBanding: YES ];
    [ scroller setDelegate: self ];
    [ self addSubview: scroller ];
}
```

```
webView = [ [ MyWebView alloc ]
    initWithFrame: [ scroller bounds ] ];
[ webView setTilingEnabled: YES ];
[ webView setTileSize: frame.size ];
[ webView setAutoresizes: YES];
[ webView setDelegate: self];
[ webView setEnabledGestures: 0xFF ];
[ webView setSmoothsFonts: YES ];
[ scroller addSubview: webView ];
return self;
}

-(id)loadURL: (NSURL *)url
{
    CGPoint zero;
    zero.x = 0;
    zero.y = 0;
    [ scroller scrollPointVisibleAtTopLeft: zero ];

    urlRequest = [ [ NSURLRequest requestWithURL: url ] retain ];
    [ webView loadRequest: urlRequest ];
}

@end
```

Листинг 7.29. Пример Web-вида и прокрутки (MyExample.h)

```
#import <CoreFoundation/CoreFoundation.h>
#import <UIKit/UIKit.h>
#import <UIKit/UITextView.h>
#import "SimpleWebView.h"

@interface MyTextView : UITextView
{
}
}
```

```
- (void)contentMouseUpInView:(id)fp8 withEvent:(struct __GSEvent *)fp12;
- (BOOL)webView:(id)fp8 shouldInsertText:(id)character
replacingDOMRange:(id)fp16 givenAction:(int)fp20;
@end

@interface MainView : UIView
{
    MyTextView *textField;
    SimpleWebView *webView;
    BOOL keyboardEnabled;
}
- (id)initWithFrame:(CGRect)frame;
- (void)contentMouseUpInView:(id)_id withEvent:(
    struct __GSEvent *)_event;
- (void)enterPressed;
- (void)dealloc;
@end

@interface MyApp : UIApplication
{
    UIWindow *window;
    MainView *mainView;
}
- (void)applicationDidFinishLaunching:
    (NSNotification *)aNotification;
@end
```

Листинг 7.30. Пример Web-вида и прокрутки (MyExample.m)

```
#import "MyExample.h"

int main(int argc, char **argv)
{
    return UIApplicationMain(argc, argv, @"MyApp", @"MyApp");
}
```

```
@implementation MyTextView
- (void)contentMouseUpInView:(id)_id
  withEvent:(struct __GSEvent *)_event
{
  [ _delegate contentMouseUpInView:(id)_id withEvent:_event ];
}

- (BOOL)webView:(id)fp8 shouldInsertText:
  (id)character
  replacingDOMRange:(id)fp16
  givenAction:(int)fp20
{
  if ( [ character characterAtIndex:0 ] == '\n')
  {
    [ _delegate enterPressed ];
    return NO;
  }

  return [ super webView:fp8 shouldInsertText:character
    replacingDOMRange:fp16
    givenAction:fp20
  ];
}
@end

@implementation MyApp
- (void)applicationDidFinishLaunching:(NSNotification *)aNotification {
  window = [ [ UIWindow alloc ] initWithContentRect:
    [ UIScreen fullScreenApplicationContentRect ]
  ];

  CGRect rect = [ UIScreen fullScreenApplicationContentRect ];
  rect.origin.x = rect.origin.y = 0.0f;

  mainView = [ [ MainView alloc ] initWithFrame: rect ];
  [ window setContentView: mainView ];
}
```

```
[ window orderFront: self ];
[ window makeKey: self ];
[ window _setHidden: NO ];
}
@end

@implementation MainView
- (id)initWithFrame:(CGRect)rect {

self = [ super initWithFrame: rect ];
if (nil != self) {
    textField = [ [ MyTextView alloc ]
initWithFrame: CGRectMake(0, 0, 320, 32) ];
    [ textField setDelegate: self ];
    [ textField setReturnKeyType: 1 ];
    [ textField scrollToMakeCaretVisible: YES ];
    [ textField setEditable: YES ];
    [ textField setText: @"http://" ];
    [ self addSubview: textField ];

    rect.origin.y = 16;
    webView = [ [ SimpleWebView alloc ] initWithFrame: rect ];
    [ self addSubview: webView ];

    keyboardEnabled = YES;
}

return self;
}

- (void)enterPressed {
    NSURL *url = [ [ NSURL alloc ] initWithString: [ textField text ] ];
    keyboardEnabled = NO;
    [ webView loadURL: url ];
}
```

```
- (void)contentMouseUpInView:(id)_id
    withEvent:(struct __GSEvent *)_event {
    if (keyboardEnabled == NO) {
        keyboardEnabled = YES;
    } else {
        keyboardEnabled = NO;
    }
}

- (void)dealloc
{
    [ self dealloc ];
    [ super dealloc ];
}

@end
```

Как это работает

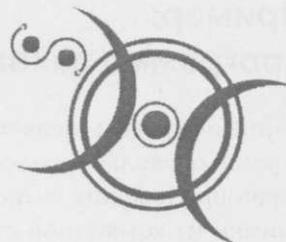
1. При порождении приложения создается `MainView` и вызывается его метод `initWithFrame`. Этот метод создает объект `UITextView`, который будет служить адресным полем, с задаваемыми свойствами клавиатуры, связанными с ним. Также создается класс `SimpleWebView`, который содержит объекты `UIWebView` и `UIScroller`, используемые для построения Web-вида. Основной вид создает объект `UIKeyboard`, но пока не добавляет его к виду.
2. Когда пользователь касается адресной строки, получает уведомления об этом событии ее метод `contentMouseUpInView`, который, в свою очередь, уведомляет своего делегата — основной вид. Основной вид включает или отключает клавиатуру путем добавления или удаления ее из основного вида.
3. Как только пользователь начинает ввод в адресной строке, для каждого вводимого символа вызывается метод `shouldInsertText`. Когда пользователь нажимает клавишу `<Return>` (обозначенную как `Go`), уведомляется метод `enterPressed` делегата.

4. Метод `enterPressed` скрывает клавиатуру и вызывает метод `loadURL` объекта `SimpleWebView`. Тем самым переустанавливается положение прокрутки в (0, 0) и продолжается загрузка новой Web-страницы или файла.

Для дальнейшего изучения

- Чтобы посмотреть на остальные творения Шона Хебера, посетите Web-узел `iApp-a-Day`, расположенный по адресу: <http://www.iappaday.com>.
- Проверьте наличие прототипов `UIWebView.h` и `UIScroller.h` в каталоге `include` пакета инструментов. Их можно найти в папке `/toolchain/sys/include/UIKit`.
- Глядя на `UIScroller.h`, поэкспериментируйте с некоторыми из дополнительных методов задания свойств, которые меняют свойства прокрутки. Что еще интересного вы сможете получить от этого класса?

ПРИЛОЖЕНИЕ



Различные приемы и способы

В этой книге были освещены только основные платформы iPhone. Но на iPhone существуют десятки меньших собственных платформ, ожидающих изучения. С помощью небольшого хакерства была доказана полезность для определенных целей некоторых из этих небольших платформ. А для выполнения таких задач, как инициация телефонного звонка или настройка вибратора iPhone, был разработан целый ряд других интересных приемов. В этом приложении рассмотрены некоторые из таких приемов и способов, которым мы не нашли место в главах этой книги.

Выполнение дампа экрана

Класс `UIApplication` содержит метод `_dumpScreenContents`, который может использоваться на iPhone для выполнения мгновенных снимков (screenshots) экрана. Этот метод записывает на диск файл `/tmp/foo_0.png`.

Чтобы выполнить дамп содержимого экрана из вашего приложения, вызовите метод `_dumpScreenContents` из вашего класса `UIApplication`:

```
[ self _dumpScreenContents: nil ];
```

Чтобы сделать мгновенный снимок экрана, вам потребуется, по крайней мере, вид основного окна с содержимым. В результате каждого вызова `_dumpScreenContents` файл записывается на одно и то же место, `/tmp/foo_0.png`, поэтому если вы делаете несколько мгновенных снимков, то каждый раз перемещайте его из этого места.

Пример: программа захвата экрана из командной строки

Данный пример делает мгновенный снимок экрана прямо из командной строки, позволяя пользователю войти в систему посредством SSH и захватить экран приложения, выполняемого в текущий момент. Выполнять мгновенные снимки из командной строки может только фирменное программное обеспечение iPhone версии 1.x. Версии 2.x и старше могут делать мгновенные снимки только из вашего приложения, поэтому если вы работаете с версией 2.x, то вам потребуется интегрировать этот код в ваш проект. Данный пример создает невидимое окно и вид, сквозь который на экране просвечивается текущее приложение. При запуске данный пример незамедлительно делает дамп содержимого экрана в файл /tmp/foo_0.png.

Чтобы скомпилировать это приложение из командной строки, воспользуйтесь пакетом инструментов следующим образом:

```
$ arm-apple-darwin9-gcc -o ScreenDump ScreenDump.m -lobjc \
    -framework UIKit -framework CoreFoundation -framework Foundation
```

Чтобы воспользоваться этим приложением, сначала скопируйте его в iPhone, используя SCP. Не забудьте при необходимости подписать его с помощью ldid:

```
$ scp ScreenDump root@iphone:/usr/bin
```

SSH в iPhone запустите из командной строки:

```
$ ssh -l root iphone
# /usr/bin/ScreenDump
```

Листинг П1 содержит соответствующий код.

Листинг П1. Пример выполнения мгновенного снимка (ScreenDump.h)

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

@interface ScreenDumpApp : UIApplication
{
    UIWindow *window;
}
- (void)applicationDidFinishLaunching:(NSNotification *)aNotification;
@end
```

```
@implementation ScreenDumpApp
- (void)applicationDidFinishLaunching:(NSNotification *)aNotification {
    CGRect rect = [ UIHardware fullScreenApplicationContentRect ];

    window = [ [ UIWindow alloc ] initWithContentRect: rect ];
    [ window orderFront: self ];
    [ window makeKey: self ];
    [ window _setHidden: YES ];
    [ window setContentView: [
        [ UIView alloc ] initWithFrame: rect ]
    ];

    printf("Dumping screen contents...\n");
    [ self _dumpScreenContents: nil ];
    [ self terminate ];
}

int main(int argc, char *argv[])
{
    NSAutoreleasePool *autoreleasePool = [
        [ NSAutoreleasePool alloc ] init
    ];
    int returnCode = UIApplicationMain(argc, argv,
        @"ScreenDumpApp", @"ScreenDumpApp");
    [ autoreleasePool release ];
    return returnCode;
}
@end
```

Как это работает

Вот как работает пример с мгновенным снимком:

1. При порождении приложения сразу же создаются объекты `UIWindow` и `UIView`. Чтобы сделать окно скрытым, используется метод `_setHidden` окна.

2. Вызывается метод экземпляра `_dumpScreenContents` класса `UIApplication`, который делает дамп содержимого экрана в файл `/tmp/foo_0.png`.
3. Затем приложение вызывает собственный завершающий метод, вполне эффективно удаляя себя.

Выполнение дампа иерархии UI

При отсутствии полнофункционального отладчика для iPhone единственным помощником для разработчиков становится метод `_dumpUIHierarchy`, предоставляемый классом `UIApplication`. Дамп UI показывает взаимные связи всех отображаемых объектов UI в типе иерархии "родитель — потомок". Например, панель навигации будет иметь в таком дампе следующий вид:

```
<dict>
  <key>CGRect</key>
  <data>
    AAAAAAAAAoEEAAKBDAABAQg==
  </data>
  <key>Children</key>
  <array>
    <dict>
      <key>CGRect</key>
      <data>
        AAAAAAAoEEAAAAAAAAAA==
      </data>
      <key>Enabled</key>
      <false/>
      <key>ID</key>
      <string>&lt;UINavigationController: 0x22fe60&gt;</string>
    </dict>
  </array>
  <key>CGRect</key>
  <data>
    AAB3QwAA+EEAAIhCAADwQQ==
  </data>
  <key>Enabled</key>
  <false/>
</dict>
```

```
<true/>
<key>ID</key>
<string>BTN Settings</string>
</dict>
</array>
<key>Enabled</key>
<true/>
<key>ID</key>
<string>&lt;UINavigationBar: 0x22f380&gt;</string>
</dict>
```

Чтобы вызвать дамп, обратитесь к методу экземпляра `_dumpUIHierarchy` класса `UIApplication`:

```
[ self _dumpUIHierarchy: nil ];
```

В `/tmp/UIDump` будет записан файл в формате XML. Все оконные методы записываются в дамп в той иерархии, в которой они размещены в текущий момент.

Вызов Safari

Иногда для отображения Web-страницы в вашем приложении удобно вызвать Safari; например, когда пользователь нажимает кнопку перехода на домашнюю страницу на экране вашего приложения с титрами. Класс `UIApplication` поддерживает метод `openURL`, который может использоваться для незаметного запуска Safari и загрузки в новом окне Web-страницы.

Чтобы воспользоваться этим, вашему приложению необходимо создать объект `NSURL`, который является основным объектом Cocoa для хранения URL. Объект `NSURL` передается в метод `openURL` приложения, где платформа приложения обрабатывает и запускает соответствующее приложение-обработчик:

```
NSURL *url;
url = [ [ NSURL alloc ] initWithString: @"http://www.oreilly.com" ];
[ self openURL: url ];
```

Инициирование телефонных звонков

Как было показано в предыдущем разделе, метод `openURL` вызывает Safari для запуска URL Web-узлов. На самом же деле происходит следующее: каждый протокол сопоставлен отдельному приложению-обработчику. Как было в случае нашего последнего демонстрационного примера, URL, начинающиеся с `http://` и `https://`, сопоставлены Safari, в связи с чем они открываются всякий раз, когда вызывается `openURL` с использованием именно этих протокольных префиксов. Метод `openURL` может использоваться не только для открытия Web-узлов в Safari, но и для осуществления телефонных звонков. Для этого применяется протокольный префикс `tel://`:

```
NSURL *url = [ [ NSURL alloc ]
    initWithString: @"tel://212-555-1234" ];
[ self openURL: url ];
```

Когда метод `openURL` используется для URL, начинающихся с `tel://`, запускается приложение телефона и автоматически совершается телефонный звонок. Обязательно протестируйте ваше приложение и убедитесь в том, что в нем нет никаких ошибок, в результате которых оно совершает недешевые международные звонки или хулиганские звонки в Белый дом.

Вибрирование

iPhone имеет встроенный вибрирующий мотор для беззвучного уведомления пользователя о новых событиях. Он управляется платформой MeCCA, являющейся частной платформой C++, используемой для низкоуровневого взаимодействия с различными устройствами, включая аудио, Bluetooth и др. Дэниель Пиблс (Daniel Peebles) написал пример низкоуровневой вибрации, который может быть обернут в приложение или вызван как независимый бинарный код.

Чтобы iPhone завибрировал, создайте экземпляр класса C++ `MeCCA_Vibrator`. Вот прототип этого класса:

```
class MeCCA_Vibrator {
public:
    int getDurationMinMax(unsigned int&, unsigned int&);
    int activate(unsigned short);
    int activate(unsigned int, unsigned short);
```

```
int activate(unsigned int, unsigned int, unsigned short);  
int deactivate( );  
};
```

Чтобы воспользоваться этим классом, сначала создайте новый экземпляр `MeCCA_Vibrator`:

```
MeCCA_Vibrator *v = new MeCCA_Vibrator;
```

Затем для управления вибрацией могут использоваться методы `activate` и `deactivate` объекта вибрации:

```
v->activate(1);  
usleep(DURATION);  
v->deactivate( );
```

При вызове функции `usleep()` укажите длительность в микросекундах, в течение которой должен работать вибратор.

Чтобы начать использовать объект `MeCCA_Vibrator`, ваше приложение должно быть подключено к платформе `MeCCA`. `MeCCA` можно подключить к вашему приложению с помощью пакета инструментов, добавив для этого аргумент `-framework MeCCA` к аргументам компилятора, описанным в главе 2:

```
$ arm-apple-darwin9-gcc -o MyApp MyApp.m -lobjc  
-framework CoreFoundation  
-framework Foundation  
-framework MeCCA
```

Чтобы добавить этот параметр в пример `make`-файла из предыдущей главы, добавьте платформу `MeCCA` в раздел флагов компоновщика, чтобы данная библиотека подключилась:

```
LDFLAGS = -lobjc  
-framework CoreFoundation  
-framework Foundation  
-framework MeCCA
```

Прозрачные виды

В главе 7 мы познакомились с классом `UICompositeImageView`, который позволяет наслаивать друг на друга несколько изображений, добавляя при этом

уровни прозрачности для создания различных эффектов. Класс `UIView` предоставляет схожую функцию `setAlpha`, позволяющую настраивать прозрачность вида:

```
[ mainView setAlpha: 0.5 ];
```

Это может пригодиться при наложении нескольких видов, например, создании полупрозрачных кнопок и панелей навигации, что позволит оставаться видимыми сквозь этот объект тексту и изображениям. Значение прозрачности варьируется от 0 до 1, где 0 делает вид абсолютно невидимым, а 1 заставляет вид полностью перекрывать собой все, что находится под ним.

Чтобы воспользоваться этим, создайте два вида с помощью перекрывающихся фреймов. Используя `setAlpha`, задайте уровень прозрачности переднего вида. Теперь добавьте к вашему управляющему виду первый вид сразу после второго. Передний вид должен быть полупрозрачным.

Переворачивание альбома в стиле Cover Flow

В главе 5 были рассмотрены трансформации `Layer Kit`, которые позволяют вращать, масштабировать и трансформировать уровень различными способами. `Layer Kit` является основой для технологии `Cover Flow` от Apple, которая используется в выборе альбомов из приложения `iPod` в альбомном режиме.

Лейтон Дункан (Layton Duncan of Polar Bear Farm), разработчик программного обеспечения для iPhone, любезно предоставил пример адаптированного для iPhone кода `CovertFlow` от Apple (из примеров инструментов Xcode). Исходный код и изображения приложения во всей их полноте можно загрузить с Web-узла "Polar Bear Farm" по адресу: <http://www.polarbearfarm.com>.

Мы немного оживили пример Лейтона и использовали фотоальбом iPhone в качестве обложек альбома. Поэтому прежде чем воспользоваться данным примером, убедитесь в том, что вы сделали несколько фотоснимков с помощью вашего iPhone.

Чтобы скомпилировать этот пример из командной строки, воспользуйтесь пакетом инструментов следующим образом:

```
$ arm-apple-darwin9-gcc -o CovertFlow CovertFlow.m -lobjc   
-framework CoreFoundation -framework Foundation -framework UIKit 
```

```
-framework QuartzCore -framework CoreGraphics ↵  
-framework GraphicsServices
```

Листинги П2 и П3 содержат соответствующий код.

Листинг П2. Пример альбома QuartzCore (CovertFlow.h)

```
#import <Foundation/Foundation.h>  
#import <CoreFoundation/CoreFoundation.h>  
#import <UIKit/UIKit.h>  
#import <UIKit/UIApplication.h>  
#import <UIKit/UIScroller.h>  
#import <UIKit/UIView-Hierarchy.h>  
#import <QuartzCore/QuartzCore.h>  
#import <QuartzCore/CAScrollLayer.h>  
#import <GraphicsServices/GraphicsServices.h>  
  
/* Количество прокрученных пикселей,  
   прежде чем следующая обложка выйдет на передний план */  
#define SCROLL_PIXELS 60.0  
  
/* Размер каждой обложки */  
#define COVER_WIDTH_HEIGHT 128.0  
  
@interface CFView : UIScroller  
{  
    BOOL beating;  
}  
- (id) initWithFrame:(struct CGRect)frame;  
- (void) mouseDragged:(GSEvent*)event;  
- (void) heartbeatCallback;  
@end  
  
@interface CovertFlowApplication : UIApplication  
{  
    UIWindow *window;  
    CFView *mainView;
```

```
CAScrollLayer *cfIntLayer;
NSMutableArray *pictures;
int selected;
}
+ (CovertFlowApplication *)sharedInstance;
- (void) jumpToCover:(int)index;
- (void) layoutLayer:(CAScrollLayer *)layer;
@end
```

Листинг ПЗ. Пример альбома QuartzCore (CovertFlow.m)

```
#import <CoreFoundation/CoreFoundation.h>
#import <Foundation/Foundation.h>
#import <UIKit/CDStructures.h>
#import <UIKit/UIWindow.h>
#import <UIKit/UIView.h>
#import <UIKit/UIView-Hierarchy.h>
#import <UIKit/UIHardware.h>
#import <UIKit/UIResponder.h>
#import <GraphicsServices/GraphicsServices.h>
#import <UIKit/UIView-Geometry.h>
#import <CoreGraphics/CGGeometry.h>
#import <UIKit/UIKit.h>
#import <QuartzCore/QuartzCore.h>
#import <QuartzCore/CALayer.h>
#import <QuartzCore/CAScrollLayer.h>
#import <QuartzCore/CAAnimation.h>
#import <QuartzCore/CATransition.h>
#import <QuartzCore/CATransaction.h>
#import <QuartzCore/CAMediaTimingFunction.h>
#import "CovertFlow.h"
#import "math.h"

static CovertFlowApplication *sharedInstance;

int main(int argc, char **argv)
{
```

```
NSAutoreleasePool *autoreleasePool = [
    [ NSAutoreleasePool alloc ] init
];
int returnCode = UIApplicationMain(argc, argv,
    @"CovertFlowApplication", @"CovertFlowApplication");
[ autoreleasePool release ];
return returnCode;
}

@implementation CFView
-(id)initWithFrame:(struct CGRect)frame
{
    self = [ super initWithFrame: frame ];
    if (nil != self) {
        [self setTapDelegate:self];
        [self setDelegate:self];
        beating = NO;
        [ self startHeartbeat: @selector(heartbeatCallback)
            inRunLoopMode:nil ];
    }
    return self;
}

-(void) mouseDragged:(GSEvent*)event
{
    if (beating == NO) {
        /* Пользователь начал пролистывать обложки.
        Начинаем момент обновления coverflow */

        beating = YES;
        [ self startHeartbeat: @selector(heartbeatCallback)
            inRunLoopMode:nil ];
    }
    [ super mouseDragged: event ];
}
}
```

```
- (void)heartbeatCallback
{
    [ [ CovertFlowApplication sharedInstance ]
      jumpToCover:(int) roundf(([ self offset ].y/SCROLL_PIXELS))
    ];

    if (! [self isScrolling] )
    {
        /* Останавливаем момент обновления,
           когда останавливается прокрутка */
        [ self stopHeartbeat: @selector(heartbeatCallback) ];
        beating = NO;
    }
}
@end

@implementation CovertFlowApplication
+ (CovertFlowApplication *)sharedInstance
{
    if (!sharedInstance) {
        sharedInstance = [ [ CovertFlowApplication alloc ] init ];
    }
    return sharedInstance;
}

- (void)applicationDidFinishLaunching:(NSNotification *)aNotification
{
    UIImageView *background;
    CGRect rect;
    NSString *file, *path;
    NSDirectoryEnumerator *dirEnum;
    int i, j;

    /* Читаем папку изображений */
    path = [ [ NSString alloc ] initWithString:
             @" /var/mobile/Media/DCIM/100APPLE" ];
```

```
pictures = [ [ NSMutableArray alloc] init ];
dirEnum = [ [ NSFileManager defaultManager ] enumeratorAtPath:
            path ];
while ((file = [ dirEnum nextObject ])) {
    if ( [ [ file pathExtension ] isEqualToString: @"THM" ] )
    {
        [ pictures addObject: [ [ NSString alloc ] initWithString:
                                [ path stringByAppendingPathComponent: file ] ] ];
    }
}
j = [ pictures count ];

window = [ [ UIWindow alloc ] initWithContentRect:
           [ UIScreen fullScreenApplicationContentRect ] ];

rect = [ UIScreen fullScreenApplicationContentRect ];
rect.origin.x = rect.origin.y = 0.0f;

sharedInstance = self;
mainView = [ [ CFView alloc ] initWithFrame: rect ];

/* Задаем # пикселей для перетаскивания до того,
   как прокрутка начнет перемещение */
[ mainView setScrollHysteresis: 64.0 ];

/* Запрещаем эффект резиновой ленты */
[ mainView setAllowsFourWayRubberBanding: NO ];

/* Это приводит к тому, что прокрутка становится более жесткой */
[ mainView setScrollDecelerationFactor: 0.9999 ];

/* Задаем прокручиваемый вид,
   чтобы перескочить к пиксельным границам */
[ mainView setGridSize:CGSizeMake(SCROLL_PIXELS, SCROLL_PIXELS) ];
```

```
[ window setContentView: mainView ];
[ window orderFront: self ];
[ window makeKey: self ];
[ window _setHidden: NO ];

/* Инициализируем уровень CovertFlow */
cfIntLayer = [ [ CAScrollLayer alloc ] initWithBounds:
    CGRectMake(0, 0, rect.size.width, rect.size.height + 128)
];
[ cfIntLayer setDelegate:self ];

/* Размещаем уровень CovertFlow в середине прокручиваемого вида */
cfIntLayer.position = CGPointMake(160, 304);
[ cfIntLayer setDelegate:self ];

/* Загружаем обложки альбомов */
for (i=0; i<j; i++) {
    NSString *filename = [ pictures objectAtIndex: i ];

    background = [ [ UIImageView alloc ] initWithFrame:
        CGRectMake(0, 0, COVER_WIDTH_HEIGHT, COVER_WIDTH_HEIGHT)
    ]
    autorelease
];
[ background setImage: [ [ UIImage alloc ]
    initWithContentsOfFile: filename ] ];
[ cfIntLayer addSublayer: [ background _layer ] ];
}

/* Задаем размер прокручиваемого вида пропорционально # обложек */
[ mainView setContentSize:
    CGSizeMake(320, ( (rect.size.height) + (SCROLL_PIXELS*j) ) )
];

/* Добавляем уровень альбома в основной уровень */
selected = 0;
```

```
[ [ mView _layer ] addSublayer:cfIntLayer ];
[ self layoutLayer: cfIntLayer ];
}

- (void) jumpToCover:(int)index
{
    if (index != selected) {
        selected = index;
        [ self layoutLayer:cfIntLayer ];
    }
}

-(void) layoutLayer:(CAScrollLayer *)layer
{
    CALayer *sublayer;
    NSArray *array;
    size_t i, count;
    CGRect rect, cfImageRect;
    NSSize cellSize, spacing, margin;
    CGSize size;
    CATransform leftTransform, rightTransform, sublayerTransform;
    float zCenterPosition, zSidePosition;
    float sideSpacingFactor, rowScaleFactor;
    float angle = 1.39;
    int x;

    size = [ layer bounds ].size;

    zCenterPosition = 60; /* Положение Z выделенной обложки */
    zSidePosition = 0; /* Положение Z по умолчанию для других обложек */
    sideSpacingFactor = .85; /* Насколько близко должны быть
                             обложки слайдов */
    rowScaleFactor = .55; /* Расстояние между основной обложкой
                           и обложками сторон */

    leftTransform=CATransform3DMakeRotation(angle,-1,0,0);
    rightTransform = CATransform3DMakeRotation(-angle,-1,0,0);
```

```
margin = NSMakeSize(5.0, 5.0);
spacing = NSMakeSize(5.0, 5.0);
cellSize = NSMakeSize (COVER_WIDTH_HEIGHT, COVER_WIDTH_HEIGHT);

margin.width += (size.width - cellSize.width * [ pictures count ] -
                spacing.width * ([ pictures count ] - 1)) * .5;
margin.width = floor (margin.width);

/* Строим массив обложек */
array = [ layer sublayers ];
count = [ array count ];

sublayerTransform = CATransform3DIdentity;
/* Задаем перспективу */
sublayerTransform.m34 = -0.006;

/* Начинаем CATransaction, чтобы все анимации
   выполнились одновременно */
[ CATransaction begin ];
[ CATransaction setValue: [ NSNumber numberWithFloat:0.3f ]
  forKey:@"animationDuration" ];
for (i = 0; i < count; i++)
{
    sublayer = [ array objectAtIndex:i ];
    x = i;

    rect.size = *(CGSize *)&cellSize;
    rect.origin = CGPointMake(0,0);
    cfImageRect = rect;

    /* Основное размещение */
    rect.origin.x = size.width / 2 - cellSize.width / 2;
    rect.origin.y = margin.height +
                    x * (cellSize.height + spacing.height);
```

```
[ [ sublayer superlayer ] setSublayerTransform:
    sublayerTransform ];
if (x < selected) /* Левая сторона */
{
    rect.origin.y += cellSize.height * sideSpacingFactor *
        (float) (selected - x - rowScaleFactor);
    sublayer.zPosition = zSidePosition - 2.0 * (selected - x);
    sublayer.transform = leftTransform;
}
else if (x > selected) /* Правая сторона */
{
    rect.origin.y -= cellSize.height * sideSpacingFactor
        * (float) (x - selected - rowScaleFactor);
    sublayer.zPosition = zSidePosition - 2.0 * (x - selected);
    sublayer.transform = rightTransform;
}
else /* Выбранная обложка */
{
    sublayer.transform = CATransform3DIdentity;
    Cover Flow-Style Album Flipping | 243

    sublayer.zPosition = zCenterPosition;
    /* Размещаем посередине прокручиваемого вида */
    [ layer scrollToPoint: CGPointMake(0, rect.origin.y -
        (([ layer bounds ].size.height - cellSize.width)/2.0))
    ];

    /* Размещаем прокручиваемый уровень в центре вида */
    layer.position =
        CGPointMake(160.0f, 240.0f + (selected * SCROLL_PIXELS));
}
[ sublayer setFrame: rect ];
}
[ CATransaction commit ];
}
@end
```

Как это работает

1. При порождении приложения вызывается его метод `applicationDidFinishLaunching`.
2. Сначала этот метод пробегает по папке альбомов iPhone и проводит опись всех изображений пиктограмм. Затем он инициализирует новое окно и подклассивизирует вид `UIScroller` под именем `CFView` в объект `mainView`. Здесь определяются различные свойства прокручиваемого вида.
3. Каждая пиктограмма фотоальбома загружается в объект `UIImage`, а затем присваивается уровню. Когда приложение завершает инициализацию, вызывается метод `layoutLayer`.
4. Метод `layoutLayer` размещает каждый уровень в виде в соответствии с его положением. Выделенный альбом переносится на передний план, а остальные поворачиваются с помощью методов `rightTransform` и `leftTransform` объектов `CATransform`.
5. Когда пользователь перемещает свой палец, вызывается метод `mouseDragged`. В результате чего выбирается новая обложка и снова вызывается `layoutLayers`.
6. Чтобы убедиться в том, что все объекты анимируются одновременно, создается `CATransaction`. Недавно выделенный альбом поворачивается к лицевой стороне экрана. Когда транзакция совершится, платформа Layer Kit выполнит все построения промежуточных изображений таким образом, что они автоматически произойдут за заданный для анимации временной интервал (0,03 с).

Предметный указатель

A

Action sheet 39, 75
Application badges 106
AppSnapp, программа 9
AppTapp, программа 9
Audio daemon 163
Audio Toolbox 176
AVController, класс 165

B

Badge 40, 297
Bundle 14

C

Categories 32
Celestial 164
CFMutableDictionary, класс 135
CGPoint, структура 114
CGRect, структура 41, 115
CGSize, структура 115
Chroot jail 7
Chroot-тюрьмы 7
Controlling view 38
Controls 60, 212, 214
Core Audio 163
Core Surface 133
Cover Flow 336

D

Data binding 84, 223, 268, 281
Data source 84
Delegate 55
Destructive button 77
Directory 14
Disclosures 88
Display region 40

E

Equalizer presets 167

F

Frame buffer 139

G

Gesture 122
GSEvent, структура 117

H

Hog mode 163

I

iNdependencee, программа 8

J

Jailbreaking 7

L

Label 87

Layer 134

M

Makefile 25

MeCCA 334

MeCCA_Vibrator 334

Messages 28

MobileTerminal 10, 14

N

Navigation bars 39

O

Objective-C 27

Orientation sensor 213

P

Pickers 212, 280

Pose 35

Preferences tables 212, 222

Program directory 14

Progress indicators 212, 239

Pwnage, программа 9

Q

Quartz Core 133

R

Repeat mode 166

S

Sample rate 167

Screen surface 135

scrollers 213, 312

Section lists 212, 267

Secure Shell 10

Segmented controls 60, 214

SimpleWebView, класс 317

Slider controls 220

Sound buffer 181, 195

SpringBoard 15

SSH 10

Status bar 39

Switch control 218

T

Table 39, 84

Text cell 227

Text views 38

Toolbar 213, 294

Transition views 66

U

UIApplication, класс 37

UIAutocorrectImageView, класс 259

UIClippedImageView, класс 259
UICompositeImageView, класс 260
UIControl, класс 214
UIDatePicker, класс 288
UIHardware, класс 41, 165
UIImage, класс 253
UIImageView, класс 258
UIKit 37
UIPickerView, класс 281
UIPreferencesTable, класс 223
UIPreferencesTableCell, класс 227
UIProgressbar, класс 240
UIProgressHUD, класс 248
UIProgressIndicator, класс 239
UISectionList, класс 267
UITable, класс 84
UITextView, класс 49
UIToolbar, класс 294
UIView, класс 214
UIWindow, класс 40

V

View 38
Volume 166

W

Web document view 213
Web views 311
Web-вид 311
Window 38, 40
Window transitions 39

X, Z

Xcode 20
ZiPhone, программа 9

A

Аккорд 119
Акселерометр 305
Аудиодемон 163
 mediaserverd 163
Аудиоочередь 170

Б

Бейдж 40, 297
 приложения 106
Буфер фрейма 139

В

Вид 38
Выборщик 212, 280

Д

Датчик положения 213
Делегат 55

Ж

Жест 122

З

Заготовки эквалайзера 167
Звуковой буфер 181, 195

И

Индикаторы прогресса 212, 239

Интерфейс 30
 Источник данных 84

К

Категория 32
 Кнопка уничтожения 77

Л

Лист действий 39, 75

М

Метка 87
 Метод 30
 _dumpScreenContents 329
 _dumpUIHierarchy 332
 applicationImageNamed 253
 cellForGroup 225
 cellForRow 225, 269
 defaultDesktopImage 254
 deviceOrientation 306
 gestureChanged 123
 gestureEnded 123
 gestureStarted 122
 heightForRow 225
 imageAtPath 254
 initWithFrame 41
 isLabelGroup 225
 kUIBarButtonButtonAction 295
 kUIBarButtonButtonInfo 295
 kUIBarButtonButtonSelectedInfo 296
 kUIBarButtonButtonTag 296
 kUIBarButtonButtonTarget 296
 kUIBarButtonButtonType 296
 numberOfColumnsInPickerView 282
 numberOfGroupsInPreferencesTable 225
 numberOfRowsInColumn 282

 numberOfRowsInTable 269
 numberOfSectionsInSectionList 269
 ringerState 165
 rowForSection 269
 setDatePickerMode 289
 setFrame 41
 setHighlightsToday 289
 setMaximumDate 289
 setMinimumDate 289
 setStaggerTimeIntervals 289
 swipe 90
 tableCellForRow 282
 tableRowSelected 271
 titleForSection 269

О

Область отображения 40
 Окно 38, 40

П

Пакет 14
 Панель:
 инструментов 213, 294
 навигации 39
 Переходные виды 66
 Переходы:
 окон 39
 уровней 144
 Поверхность экрана 135
 Полиморфизм 59
 Преобразование уровней 154
 Привязка данных 84, 223, 268, 281
 Прокрутка 213, 312

Р

Раскрытия 88
 Режим повторения 166

С

Событие мыши 119
 mouseDown 120
 mouseDragged 121
 mouseEntered 121
 mouseExited 121
 mouseMoved 121
 mouseUp 120
Сообщение 28
Списки разделов 212, 267
Строка состояния 39, 102, 124

Т

Таблица 39, 84
 предпочтений 212, 222
Текстовая ячейка 227
Текстовые виды 38

У

Управляющий вид 38
Уровень 134
 громкости 166

Ч

Частота звуковой дорожки 167

Э

Эквалайзер 167
Элемент управления 60, 212, 214
 переключающий 218
 сегментированный 214
 сегментный 60
 слайдер 220

Готовится к изданию на русском языке

iPhone SDK. Разработка приложений

Джонатан Зdziarski



iPhone. Разработка приложений с открытым кодом 2-е издание



С появлением и широким распространением iPhone программисты с энтузиазмом начали создавать приложения для этого устройства. При этом многие предпочитают использовать не SDK и инструменты, предлагаемые Apple, а инструментарий, созданный сообществом iPhone Dev team для разработки приложений с открытым кодом. Книга посвящена последней версии инструментария с открытым кодом, обновленного для программного обеспечения iPhone 2.x и iPhone 3G. В ней доступным языком объясняется, как создавать приложения с помощью Objective-C и iPhone API. Автор книги – человек, который взломал код iPhone и создал первое полнофункциональное приложение с помощью инструментария с открытым кодом, — включил в книгу детальное описание приемов программирования, а также примеры работы с графикой и звуком, программирования игр с использованием интерфейсов CoreSurfaces и CoreImage, работы с iTunes, использования сенсоров и др.

С помощью инструментария с открытым кодом и этой книги вы научитесь создавать приложения для iPhone, которые смогут:

- Отображать строку состояния, таблицы предпочтений и другие стандартные элементы пользовательского интерфейса iPhone
- Воспроизводить файлы или генерировать звуки
- Читать и записывать текстовые файлы и файлы в формате HTML, включая страницы из Интернета
- Управлять элементами экрана, например, полосами прокрутки
- Реагировать на изменения пространственной ориентации iPhone и др.

«Замечательная книга не только для профессионалов! Даже не зная объектно-ориентированное программирование, но прочитав эту книгу, вы при желании быстро добьетесь результатов и создадите приложения, которые смогут конкурировать с приложениями, разработанными Apple».

*Джош Конмент (Josh Content),
разработчик iPhone*

Джонатан Зdziarski (Jonathan Zdziarski), более известный в среде хакеров как «NerveGas», сыграл ключевую роль при исследовании возможностей iPhone для разработки открытого программного обеспечения. Именно он разработал первое приложение, которое использует все основные функции API iPhone. Джонатан также занимается созданием интеллектуальных спам-фильтров. Является автором книг «iPhone Forensics», «iPhone SDK. Разработка приложений».

ISBN 978-5-9775-0397-6



9 785977 503976



БХВ-Петербург
190005, Санкт-Петербург,
Измайловский пр., 29
E-mail: mail@bhv.ru
Internet: www.bhv.ru
Тел.: (812) 251-42-44
Факс: (812) 320-01-79