+ Вводный курс по Objective-C

iPhone SDK

Разработка приложений

O'REILLY® bhv®

Джонатан Здзиарски

iPhone SDK Application Development

Jonathan Zdziarski

O'REILLY®

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

Джонатан Здзиарски

iPhone SDK

Разработка приложений

Санкт-Петербург
«БХВ-Петербург»
2013

УДК 681.3.06
ББК 32.973.26-018.2
З-46

Здзиарски Дж.

З-46 iPhone SDK. Разработка приложений: Пер. с англ. — СПб.: БХВ-Петербург, 2013. — 512 с.: ил.

ISBN 978-5-9775-0178-1

Книга посвящена разработке мобильных приложений и игр для iPhone и iPod Touch с использованием SDK Apple. Описаны основные этапы процесса разработки, язык Objective-C, а также все основные библиотеки SDK и примеры их использования. Рассмотрены проектирование и создание элементов пользовательского интерфейса с помощью Interface Builder и библиотеки UI Kit, создание элементов управления приложением, работа с уровнями и 3D-преобразования с использованием Core Graphics и Quartz Core, микширование и воспроизведение звуковых файлов с помощью AVFoundations, управление сетевыми возможностями с помощью платформы CFNetwork, использование библиотеки Core Location для осуществления взаимодействия с GPS iPhone и многое другое. Материал сопровождается большим количеством примеров.

Для программистов

УДК 681.3.06
ББК 32.973.26-018.2

Группа подготовки издания:

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

Подписано в печать 07.03.13.
Формат 70×100 1/16. Печать офсетная. Усл. печ. л. 41,28.
Доп. тираж 1000 экз. Заказ №
"БХВ-Петербург", 190005, Санкт-Петербург, Измайловский пр., 29.

Первая Академическая типография "Наука"
199034, Санкт-Петербург, 9 линия, 12/28

Authorized translation of the English edition of iPhone SDK. Application Development, ISBN: 978-0-596-15405-9, 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 iPhone SDK. Application Development, ISBN: 978-0-596-15405-9, © 2009 Jonathan Zdziarski. Все права защищены. Перевод опубликован и продается с разрешения O'Reilly Media, Inc., собственника всех прав на публикацию и продажу издания.

ISBN 978-0-596-15405-9 (англ.)
ISBN 978-5-9775-0178-1 (рус.)

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

Оглавление

Предисловие **1**

Введение **5**

Глава 1. Начало работы с iPhone SDK **11**

Анатомия приложения 11
За кулисами Xcode 14
Установка iPhone SDK 17
Что вам потребуется 18
Ключ разработчика Apple 18
iPhone 18
Симулятор iPhone 19
Загрузка и установка iPhone SDK 20
Инициализация iPhone 21
Построение и установка приложений 23
Модель — Представление — Контроллер 24
Шаблоны приложений 25
Содержимое проекта Xcode 26
Прототипы 28
Добавление библиотек 29
Установка активного SDK 31
Построение приложения 31
Установка приложения 31
Переход к Objective-C 32
Обмен сообщениями 32
Объявление классов и методов 34
Тип данных *id* 35
Импорт 35
Объявление интерфейсов 35
Методы 35

Реализация 36
Свойства 38
Протоколы 39
Категории 40
Подстановка 43
Дополнительные источники 44

Глава 2. Interface Builder: Xcode GUI для графических пользовательских интерфейсов 45

Окна, представления и контроллеры представлений 46
Существующие шаблоны 47
Новые шаблоны 48
Элементы пользовательского интерфейса 49
Контроллеры 51
Представления данных 51
Ввод данных и значения 52
Окна, представления и панели 52
Inspector 52
Разработка пользовательского интерфейса 52
Окно 53
Контроллер представлений 53
Представления 54
Соединение представлений 55
Добавление связывания в код 56
Удаление Interface Builder из проекта 57

Глава 3. Введение в UI Kit 59

Основные элементы пользовательского интерфейса 60
Окна и представления 63
Создание окна и представления 63
Отображение вида 65
Самое бесполезное приложение: HelloView 65
Как это работает 68
Порождение от класса *UIView* 68
Традиционное бесполезное приложение: HelloWorld 71
Как это работает 73
Контроллеры представлений 74
Создание контроллера представлений 75
Загрузка из Interface Builder 77

Изменение ориентации 78
Удаление контроллера представлений 79
HelloWorld в стиле контроллера представлений: ControllerDemo 79
Как это работает 83
Для дальнейшего изучения 84
Текстовые представления 84
Создание текстового вида 85
Редактирование 86
Поля 86
Шрифт и размер 87
Цвет текста 88
Цвета из библиотеки Core Graphics 89
Задание содержимого 90
Отображение HTML 91
Чтение исходного кода Web-страницы: SourceReader 92
Как это работает 96
Для дальнейшего изучения 97
Панели навигации и контроллеры 97
Создание контроллера навигации 98
Свойства контроллера навигации 99
Задание заголовка 100
Кнопки, стили и действия 100
Стиль панели навигации 102
Добавление сегментированного элемента управления 103
Добавление панели инструментов 105
Текстовые кнопки и кнопки с изображением 105
Системные кнопки 106
Собственные кнопки представлений 107
Создание панели инструментов 107
Изменение размеров 108
Стиль панели инструментов 108
Страничная навигация: PageDemo 109
Как это работает 115
Для дальнейшего изучения 116
Анимации переходов 117
Создание перехода 118
Функция распределения во времени 118
Типы анимации 118
Длительность 119
Прикрепление перехода 120

Переходы с переворачиванием страниц: FlipDemo 120
Как это работает 126
Для дальнейшего изучения 127
Листы действий и предупреждения 127
Предупреждения 128
Листы действий 129
Отмена листа действий 130
Конец света (с подтверждением): EndWorld 131
Как это работает 136
Для дальнейшего изучения 137
Табличные представления и контроллеры 137
Создание таблицы 138
Наследование класса *UITableViewController* 138
Ячейки таблицы 140
Отображаемый текст 141
Выравнивание 141
Шрифт и размер 142
Цвет текста 142
Изображения 143
Стиль выбора 144
Метки 144
Раскрытия 145
Реализация множественного выбора 146
Редактирование и действие "провести, чтобы удалить" 147
Перезагрузка таблицы 148
Простейший проводник файлов: TableDemo 149
Как это работает 156
Для дальнейшего изучения 157
Манипуляции строкой состояния 158
Скрытие строки состояния 158
Стиль строки состояния 158
Ориентация строки состояния 159
Бейджи приложения 160
Отображение бейджа приложения 160
Удаление бейджа приложения 161
Для дальнейшего изучения 161
Сервисы приложения 161
Приостановка и возобновление 162
Прекращение работы программы 163
Вызов Safari 163
Инициация телефонных звонков 164

Глава 4. События множественных касаний и геометрия 165

Введение в геометрические структуры 165
Структура *CGPoint* 166
Структура *CGSize* 166
Структура *CGRect* 167
Включение и пересечение 167
Обнаружение границы и центра 168
Обработка событий множественных касаний 168
Уведомления *UITouch* 169
Объект *UIEvent* 171
Обработка событий 171
Пример: счетчик касаний 173
Пример: коснитесь и перетащите 174
Обработка множественного касания 175
Отслеживание пинчей: PinchMe 176
Отслеживание перетаскивания значков: TouchDemo 179
Как это работает 185
Для дальнейшего изучения 185

Глава 5. Программирование уровней с использованием Quartz Core 187

Понятие уровней 187
Иерархия уровней 189
Размер и смещение 190
Упорядочивание и отображение 190
Визуализация 191
Преобразования 192
Анимация уровней 192
Преобразования уровней 193
Развлечение с уровнями: BounceDemo 195
Как это работает 201
Для дальнейшего изучения 202

Глава 6. Создание шума: Audio Toolbox и AVFoundation 203

Библиотека AVFoundation 204
Звуковой проигрыватель 205
Свойства проигрывателя 206
Воспроизведение звуков 207

Методы-делегаты 207
Снятие измерений 208
Создание волюметра: AVMeter 209
Как это работает 220
Для дальнейшего изучения 220
Аудиосервисы 220
Как это работает 222
Аудиоочереди 222
Структура аудиоочереди 223
Подготовка аудиовывода 225
Звуковые буферы 227
Функция обратного вызова 228
Уровень громкости 230
Пример: проигрыватель PCM 230
Как это работает 236
Для дальнейшего изучения 236
Запись звука 237
Структура аудиоочереди 238
Подготовка аудиоввода 240
Звуковые буферы 241
Функция обратного вызова 243
Осуществление доступа к необработанным данным 244
Запись в файл 245
Пример: магнитофон 246
Как это работает 249
Для дальнейшего изучения 250
Вибрация 250

Глава 7. Сетевое программирование с CFNetwok 251

Программирование сокетов 252
Типы сокетов 252
Объект *CFSocket* 253
Создание новых сокетов 253
Создание сокетов из существующего сокета 255
Функции сокетов 256
Разрешение/запрет обратных вызовов 257
Отправка данных 257
Обратные вызовы 258
CFSocketContext 259

Потоки сокетов....260
Потоки чтения....260
Потоки записи....263
Пример с *CFSocket*: сервер анекдотов....265
Для дальнейшего изучения....268
Интерфейсы *CFHTTP* и *CFFTP*....269
CFHTTP....269
CFFTP....271
Для дальнейшего изучения....272

Глава 8. Определение местоположения: Core Location....273

Менеджер Core Location....274
Параметры запроса....275
Выполнение запроса....277
Получение обновлений....277
Завершение запроса....279
Обработка ошибок....280
Определение местоположения: WhereYouAt....280
Как это работает....284
Для дальнейшего изучения....284

Глава 9. Библиотеки для работы с адресной книгой....285

Доступ к адресной книге....286
Функции адресной книги верхнего уровня....287
Выполнение запросов к адресной книге....287
Создание записей....288
Работа с записями....289
Запись свойств....291
Многозначные свойства....291
Запись многозначных записей....294
Работа со словарями....294
Данные изображения....295
Для дальнейшего изучения....296
Address Book UI....296
Представления для отображения контактов....296
Выборщики контактов....297
Методы-делегаты....298
Для дальнейшего изучения....299

Глава 10. Проектирование UI Kit для опытных 301

Элементы управления 303
Базовый класс *UIControl* 304
Свойства 304
Уведомления о событиях 305
Сегментированные элементы управления 308
Создание элемента управления 308
Добавление сегментов 309
Заголовки сегментов 310
Изображения 310
Мгновенные щелчки 311
Инициализация сегмента по умолчанию 311
Отображение элемента управления 311
Считывание элемента управления 311
Переключатели 312
Создание элемента управления 312
Альтернативные цвета 313
Отображение элемента управления 313
Расположение переключателя 314
Полосы прокрутки 314
Создание элемента управления 315
Отображение элемента управления 316
Считывание элемента управления 317
Текстовые поля 317
Параметры стиля 318
Визуализация подмен 319
Методы-делегаты 320
Уведомления 322
Прокрутка текстовых полей 322
Кнопки 324
Создание элемента управления 324
Отображение элемента управления 326
Визуализация подмен 327
Страницы 327
Создание элемента управления 328
Отображение элемента управления 328
Уведомления 329
Для дальнейшего изучения 329
Таблицы настроек 330
Создание таблицы настроек 330
Создание подклассов контроллера представления таблицы 332

Инициализация таблицы 333
Ячейки таблицы настроек 334
Элементы управления 335
Текстовые поля 336
Отображение таблицы настроек 336
Пример таблицы настроек: ShootStuffUp 337
Как это работает 345
Для дальнейшего изучения 346
Списки разделов 346
Создание списка разделов 347
Добавление индексной панели 349
Отображение списка разделов 350
Улучшенный проводник файлов: TableDemo 350
Как это работает 360
Для дальнейшего изучения 361
Индикаторы прогресса и активности 361
Класс *UIActivityIndicatorView*: то, что вертится 362
UIProgressView: когда вращающиеся штучки не подходят 363
Индикаторы сетевой активности 364
Для дальнейшего изучения 365
Изображения 365
Объект изображения 365
Работа с файлами (статические методы) 366
Работа с URL и необработанными данными (статические методы) 366
Работа с Core Graphics (статические методы) 367
Работа с файлами (методы экземпляров) 367
Работа с URL и необработанными данными (методы экземпляров) 367
Работа с Core Graphics (методы экземпляров) 368
Отображение изображения 368
Вывод на экран узоров 369
Ориентация 369
Размер изображения 370
Развлечение с изображениями и узорами: ImageFun 370
Изображение с представлением: *UIImageView* 374
Выборщики изображений 376
Источники изображений 376
Редактирование изображений 376
Выбор изображений 377
Свойства клавиатуры 377
Стиль клавиатуры 378

Внешний вид клавиатуры 379
Клавиша возврата 380
Автоматическое выделение прописными буквами 380
Автозамена 381
Защищенный ввод текста 381
Выборщики 382
Создание выборщика 382
Получение свойств выборщика 382
Источник данных выборщика 383
Отображение выборщика 384
Считывание выборщика 384
Выбор типа вашего носа: NosePicker 384
Как это работает 391
Для дальнейшего изучения 392
Выборщик даты и времени 392
Создание выборщика даты и времени 392
Режимы *DatePicker* 393
Временные интервалы 393
Диапазоны дат 393
Отображение выборщика даты 394
Считывание даты 394
Пример: выборщик Дня независимости 395
Как это работает 400
Для дальнейшего изучения 400
Панели вкладок 401
Контроллеры панели вкладок 401
Создание контроллера панели вкладок 402
Создание коллекции 402
Настройка свойств кнопок 403
Создание контроллера панели вкладок 404
Отображение контроллера панели вкладок 404
Настраиваемые кнопки 404
Навигация 405
Делегированные действия 405
Еще один способ реализации текстовой книги: TabDemo 406
Как это работает 410
Для дальнейшего изучения 410
Показания сенсоров и информация об устройстве 411
Считывание ориентации 411
Считывание информации об устройстве 412

Считывание показаний акселерометра 412
Отслеживание перемещений 413
Датчик приближения 414
Для дальнейшего изучения 415
Представления прокрутки 415
Создание представления прокрутки 415
Свойства 416
Методы-делегаты 417
Прокрутка метеорологической карты: BigImage 419
Как это работает 422
Для дальнейшего изучения 423
Web-представления 423
Создание Web-представления 424
Отображение Web-представления 424
Загрузка содержимого 424
Навигация 425
Методы-делегаты 425
Поисковая программа Google: WebDemo 426
Как это работает 431
Для дальнейшего изучения 431

Глава 11. Параметры приложения 433

Словари и списки свойств 433
Создание словаря 434
Управление ключами 434
Запись списка свойств 434
Считывание списков свойств 435
Для дальнейшего изучения 435
Блок настроек приложения 435
Добавление ключей 436
Групповые разделители 437
Текстовые поля 437
Изменение значений переключателей 439
Полосы прокрутки 440
Многозначные поля 442
Дочерние панели 443
Считывание значений блока настроек приложения 444
Для дальнейшего изучения 444

Глава 12. Cover Flow 445

Программирование Cover Flow в SDK: CovertFlow 446
Как это работает 455
Для дальнейшего изучения 456

Глава 13. Перелистывание страниц 457

Пример перелистывания страниц: PageControl 458
Как это работает 467
Для дальнейшего изучения 467
Класс *PageScrollView* для нескольких представлений 468
Как это работает 475

Глава 14. Библиотека Media Player 477

Контроллеры видеопроигрывателя 477
Свойства 478
Элементы управления 478
Форматное соотношение 479
Цвет фона 479
Начало и остановка воспроизведения видеофильма 479
Уведомления 480
Для дальнейшего изучения 481

Предметный указатель 482

Предисловие

За долгое время моего участия в сообществе взломщиков iPhone меня часто спрашивали о том, что я думаю об iPhone SDK. Я отблагодарю тех из вас, кто приобрел эту книгу, ответив на данный вопрос. В двух словах, iPhone SDK от Apple предоставляет определенную высокоуровневую функциональность, позволяющую разобраться во всем остальном беспорядке. Глубоко под удобным покрывалом от SDK скрыто сильно дезорганизованное и плохо спроектированное множество библиотек, хотя некоторые из них являются весьма действенными в тех областях, в которых SDK таковым не является. Без сомнения, SDK вполне подходит для написания высококачественных функциональных приложений для AppStore (если бы это было не так, я упомянул бы об этом). Предоставляемые SDK интерфейсы достаточно неплохо написаны с точки зрения большинства хороших разработчиков, создающих качественное программное обеспечение, однако большинство из них даже не подозревает о той функциональности, которая *не* доступна им. Для тех же, кто уже попробовал себя в мире открытого кода, iPhone SDK все еще остается предметом споров.

Если вы незнакомы с политикой, окружающей SDK, то должны знать о том, что существуют два множества интерфейсов: предоставляемые SDK и используемые Apple. Хотя между ними и есть некоторое перекрытие, тем не менее, я описал множество классов и библиотек, о которых вы никогда не слышали, в своей книге "iPhone. Разработка приложений с открытым кодом"[1]. Вы никогда не слышали о них потому, что они недоступны в SDK. Многие из нас на заре появления сообщества взломщиков iPhone открывали их путем прямого взлома операционной системы iPhone. За долгие недели выгрузки таблиц символов, классов и экспериментирования с пробными версиями, содержащими ошибки, мы выделили геном пакета пользовательского интерфейса iPhone, а также многих других библиотек, включая те из них, которые на сегодняшний день являются частными. Это именно те низкоуровневые API, которые разработчики используют при создании программного

---

[1] Здзиарски Д. iPhone. Разработка приложений с открытым кодом. 2-е изд. — СПб.: БХВ-Петербург, 2009.

обеспечения для iPhone с помощью связки инструментов с открытым кодом, и именно те низкоуровневые API, преимущества которых, как выясняется, используют многие приложения Apple для реализации такой функциональности, которую не могут реализовать использующие SDK приложения.

Эти низкоуровневые API дают преимущество разработчикам, использующим связку инструментов с открытым кодом, перед разработчиками, использующими SDK, и, по моему мнению, предлагают гораздо лучшую среду разработки, нежели SDK от Apple. Многие библиотеки на этом устройстве были без особой шумихи приватизированы, сделав тем самым свою функциональность недоступной для разработчиков, пишущих для AppStore. Совершенно не случайно большая часть этой функциональности оказалась ключевой для создания приложений, которые Apple считает конкурирующими с ее собственными встроенными приложениями. Самым большим ограничением является запрет библиотеки Core Surface, которая дала бы использующим SDK разработчикам возможность выводить необработанные пикселы непосредственно на уровень экрана и использовать графические ускорители. Без этой библиотеки у вас возникнут трудности с производительностью приложений, которым требуется 2D-визуализация, например, видеопроигрыватели, пишущие видеопроигрыватели или высокопроизводительные 2D-игры, как, например, мой бесплатный эмулятор Nintendo. Кроме того, это ключевая библиотека, требующаяся для написания таких приложений, как Flash или Java™ с любым уровнем производительности. Другое множество API, которого вам будет не хватать, — это возможность взаимодействия с музыкой iTunes. Вот почему SDK-версия Tap Tap Revolution Найта Тру (Nate True) больше не выбирает песни из вашей библиотеки iTunes, и почему отличные музыкальные приложения, подобные SynchStep (которое воспроизводит музыку в ритме ваших шагов), вы можете найти только в хранилищах сторонних фирм. Даже такая простейшая функциональность, как возможность работы в фоне или отображение значков строки состояния, доступна только при использовании API, которые запрещены для приложений, публикуемых в AppStore. Нет необходимости говорить, что компилятор открытого кода для iPhone позволяет вам делать то, что SDK делать не разрешает, что, как считают некоторые, сделано для того, чтобы Apple всегда имела конкурентное преимущество на рынке программного обеспечения для iPhone.

С другой стороны, SDK предоставляет то, в чем открытый код никогда не преуспевал: потенциальная возможность получения огромного дохода. Похоже, разработчики могут преодолеть свою неприязнь к политике Apple, приняв во внимание то неприличное количество денег, которое можно получить от такой большой аудитории потенциальных покупателей, которая име-

ется, к примеру, у iTunes. AppStore дает денежное вознаграждение тем новаторам, которые пожелают к нему присоединиться. Потенциальный доход, предоставляемый AppStore, дает разработчикам существенное преимущество над сообществом разработчиков открытого кода, даже если ваше приложение окажется слегка ограниченным в своей функциональности.

Исключительно с технической точки зрения компилятор с открытым кодом может создавать приложения, используя как интерфейсы SDK, так и низкоуровневые "частные" интерфейсы, в зависимости от того, какое множество заголовков вам нужно использовать. То же самое верно и относительно Xcode: частные, недокументированные интерфейсы могут быть легко импортированы в ваш проект путем простого указания SDK правильных заголовков. Это дает вам четыре возможных подхода к разработке приложений.

Но самое важное во всем этом заключается в том, что если вы хотите публиковать свои приложения в AppStore, то вы обязаны играть по правилам Apple. Apple не примет к публикации приложение, которое использует частные интерфейсы или библиотеки. По слухам, Apple отклонила даже приложения-фонарики, которые переступили черту и имели смелость попытаться самостоятельно настроить яркость экрана. Если вы являетесь коммерческим разработчиком или разрабатываете программное обеспечение для развертывания внутри своей корпорации, то для вас существует только один путь — использовать санкционированный API, описываемый в этой книге. Но если вы читаете эту книгу как энтузиаст-разработчик открытого кода и рассматриваете свой код скорее как некую форму искусства, то, возможно, вас больше заинтересует создание программного обеспечения так, как оно и должно создаваться — без каких-либо ограничений и песочниц. В таком случае я рекомендую вам рассмотреть вопрос об использовании не только тех API, описание которых вы найдете в данной книге, но и множества существующих недокументированных API и библиотек. Сообщество разработчиков с открытым кодом было первым, кто создал общедоступный компилятор и интернет-хранилище программного обеспечения для iPhone-сообщества, которое с радостью принимает великолепные полнофункциональные приложения.

На сегодняшний день мир разработки для iPhone очень страдает от такого разделения. Оба лагеря разработчиков растут, но при этом противостоят друг другу. Многие разработчики разочаровываются тем количеством ограничений, которые имеются в SDK, и утверждают, что они несовместимы с популярными лицензиями на открытый код (такими как GPL), или зачастую кричат о том, что во взломе iPhone не было бы необходимости, если бы Apple открыла свое великолепное устройство.

Я очень хочу, чтобы Apple действительно открыла операционную систему iPhone вместо того, чтобы продолжать создавать такие недоверчивые и явно монополистские ограничения в SDK. Хотя iPhone и является, по сути, самым революционным устройством в истории, тем не менее, Apple подвергается риску нанести убытки как разработчикам, так и потребителям. Простой факт, состоящий в том, что я могу приобрести устройство с гораздо меньшими преградами, нежели iPhone многое говорит о страстном желании Apple все контролировать. Эта же непробиваемая позиция волнует и многих разработчиков, пытающихся разрабатывать программное обеспечение для этой великолепной платформы.

По моему мнению, творчество и инновации разумных разработчиков не должны становиться прихотью производителей устройств. Код является формой самовыражения для многих, и навязывать цензуру на свободу самовыражения — это только добиваться запрета на инновации, если они исходят не из лагеря Apple. Несмотря на все это, я продолжаю благоговеть перед впечатляющими возможностями продуктов Apple и аплодировать их инновациям. Хорошо продуманным является не только SDK, но и последняя версия Objective-C — самый элегантный и ориентированный на разработчика язык, который я встречал до сегодняшнего дня. Apple способна создавать великолепные вещи, и практически все, что касается iPhone, потрясающе. Я надеюсь, что Apple продолжит развивать свой успех, становясь более открытой творчеству и не подавляя инновации, приходящие извне.

Джонатан Здзиарски.

Январь, 2009

Введение

Разработчики корпоративных систем в марте 2008 года с радостью встретили новость, в которой Apple анонсировала выпуск официального SDK. Долгожданная среда разработки, наконец, позволила разработчикам проектировать коммерческие приложения для iPhone и предоставила канал распространения, способный достичь каждого отдельно взятого пользователя iPhone. В данной книге освещен пакет SDK, официально одобренный Apple, и последующие API, используемые для разработки приложений, специально предназначенных для AppStore.

SDK от Apple стал огромным прогрессом в разработке мобильного программного обеспечения и предоставил фантастические возможности для увеличения своего благосостояния. Как разработчик, использующий SDK, вы имеете прямой доступ к миллионам конечных пользователей, которые получают возможность моментального приобретения вашего продукта. Последующее долгожданное снятие NDA от Apple привело к еще большему усилению энтузиазма по отношению к этому фантастическому устройству и такой бизнес-модели. Нет никаких сомнений в огромном потенциале iPhone SDK для появления новых идей и получения прибыли.

Однако энтузиазм должен сдерживаться реалистичными ожиданиями. Как разработчик для iPhone, вы будете проектировать на платформе, которая все еще является достаточно закрытой. Ваши приложения будут выполняться в ограниченной среде — песочнице (sandbox), чтобы не допускать доступ к определенным ресурсам, а Apple запретит вам использование множества частных API, имеющих доступ к более мощным ресурсам устройства. Вам необходимо знать о подобных ограничениях среды, чтобы вы не тратили впустую время на код, исходя из некорректных предположений о том, что вы можете делать, а что нет.

Хотя многие видят в SDK только пару сдерживающих их порывы наручников, тем не менее, он является достаточно мощной платформой для написания игр и приложений высокого качества. SDK представил простые в использовании объекты, лежащие поверх более сложных низкоуровневых

платформ. Это сделало кодирование таких элементов, как пользовательский интерфейс, запросов глобального местоопределения и даже настройку связок гораздо менее трудоемкими задачами, чем при использовании других сред разработки. В связи с этим разработчики могут сфокусироваться на более важных аспектах проекта. Всего лишь с помощью нескольких строк вы можете создать множество разновидностей пользовательского интерфейса, работать с анимациями 3D и микшировать звук. Эта книга познакомит вас с парадигмой разработки для iPhone и расскажет вам о платформах, являющихся ключевыми в проектировании полнофункционального программного обеспечения для iPhone.

Аудитория этой книги

Эта книга предназначена как для начинающих, так и для опытных разработчиков, пишущих для iPhone. Чтобы эта книга была для вас максимально полезна, вы должны иметь предварительное знакомство с кодированием. Среда разработки iPhone использует Objective-C, с которым вы сразу же и познакомитесь. Положительным моментом является то, что вы в своих приложениях также можете использовать C и C++, поэтому все, кто имеет соответствующие познания, смогут быстро перейти к использованию Objective-C. Данная книга не является полноценным руководством по Objective-C, однако она поможет вам получить представление о нем с помощью небольшого вводного курса и бессчетного количества примеров кода.

По мере чтения этой книги имейте в виду, что существует и другая сторона данного устройства, которая не затрагивается в этой книге. Множество объектов низкого уровня и платформы запрещены в SDK, однако используются сообществом взломщиков iPhone. В этой книге вы не найдете ни один из этих несанкционированных API за исключением нескольких явно обозначенных примеров, чтобы не вводить вас в заблуждение по поводу того, что вы можете, а что не можете использовать. Если вы используете SDK для написания приложений, которые будете использовать для своих личных нужд, или же если вы хотите получить более глубокое представление о том, как работает iPhone на низком уровне, то, возможно, захотите эту книгу дополнить книгой "iPhone. Разработка приложений с открытым кодом"[1]. В связке эти книги дадут вам полное представление о том, что вы можете, а что не можете делать,

[1] Здзиарски Д. iPhone. Разработка приложений с открытым кодом. 2-е изд. — СПб.: БХВ-Петербург, 2009.

а также о тех видах функциональности, которые неизбежно будут соперничать в приложениях, написанных сообществом взломщиков.

Структура книги

В *главе 1* рассказывается о том, как приступить к работе с iPhone SDK и как строить и устанавливать простейшие приложения.

В *главе 2* вы познакомитесь с Interface Builder, инструментом WISIWIG, используемым для проектирования патентованных iPhone пользовательских интерфейсов.

В *главе 3* вы узнаете о платформе UI Kit и научитесь проектированию основных элементов пользовательского интерфейса.

В *главе 4* рассказывается об обработке событий и основных геометрических структурах.

В *главе 5* показано, как создавать и управлять слоями и преобразованиями с помощью Core Graphics и Quartz Core.

В *главе 6* вы узнаете, как с помощью AVFoundation микшировать и воспроизводить звуковые файлы, и как с помощью платформы Audio Toolbox записывать и воспроизводить потоки цифрового звука.

В *главе 7* рассказывается о сетевом программировании с помощью платформы CFNetwork.

Глава 8 познакомит вас с платформой Core Location и покажет, как работать с GPS iPhone.

В *главе 9* рассказывается об API адресной книги (Address Book) и о том, как запрашивать и отображать контакты.

В *главе 10* освещены более сложные классы UI Kit.

В *главе 11* вы узнаете, как считывать и записывать настройки приложений, а также как работать с листами свойств.

В *главе 12* показано, как создавать каталог альбомов в стиле Cover Flow.

В *главе 13* вы увидите, как переворачивать страницы и как переключаться между множественными видами наподобие страниц в книге.

В *главе 14* рассказывается о том, как добавить видеопроигрыватель в ваше приложение.

Используемые в этой книге обозначения

В данной книге используются следующие соглашения об обозначениях:

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

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.

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**.

Примеры кода в этой книге написаны и проверены на iPhone SDK от Apple версий 2.1 и 2.2. По мере выпуска Apple новых версий возможно появление небольших изменений в API. Перед тем как приступить к какой-либо новой разработке, обязательно сверьтесь с примечаниями, прилагаемыми к новым версиям SDK от Apple, и с руководством iPhone OS Programming Guide от Apple.

Юридический отказ от ответственности

Технологии, рассматриваемые в этой публикации, ограничения в этих технологиях, которые сами технологии и владельцы содержимого стремятся наложить, а также законы, фактически ограничивающие использование этих технологий, постоянно меняются. Таким образом, некоторые проекты, описанные в этой публикации, могут не работать, могут привести к непредвиденному повреждению оборудования или систем, на которых они используются, или могут быть несовместимы с действующими законами или пользовательскими соглашениями. Вы используете эти проекты на свой страх и риск, а O'Reilly Media, Inc. отказывается от ответственности за любой ущерб или любые расходы, возникающие в связи с их использованием. В любом случае вам следует самим позаботиться о том, чтобы ваше использование этих проектов не нарушало никаких действующих законов, включая законы об авторском праве.

Мы будем рады услышать вас

У этой книги имеется Web-узел, на котором приведены список опечаток, примеры и другая дополнительная информация:

http://www.oreilly.com/catalog/9780596154059

Комментарии и технические вопросы, относящиеся к этой книге, отправляйте по следующему адресу электронной почты:

bookquestions@oreilly.com

Более подробную информацию о наших книгах, конференциях, ресурсных центрах (Resource Centers) и сети O'Reilly Network можно найти на нашем Web-узле по адресу: **http://www.oreilly.com**.

Благодарности

Выражаю особую благодарность Лейтону Дункану (Layton Duncan), Брайану Вайтману (Brian Whitman) и многим другим, кто делился со мной своими забавными находками. Также я хотел бы поблагодарить Джонатана Холла (Jonathan Hohle), Далласа Брауна (Dallas Brown), Бреда О'Херни (Bred O'Hearne) и Джона Драпера (John Draper) за их техническое рецензирование данной книги и предложения по ее улучшению. Наконец, спасибо моей жене за то, что не убила меня во сне за мое помешательство на этой книге.

Глава 1

Начало работы с iPhone SDK

Если вы новичок в мире Mac, то будете удивлены тем фактом, что приложения в этом мире 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 их удаление осуществляется путем перетаскивания в корзину. В этой главе будет объяснена структура приложений iPhone. Кроме того, вы познакомитесь и начнете использовать iPhone SDK, изучите Apple IDE, известный под названием Xcode, а также узнаете, как устанавливать приложения на ваш iPhone. В конце 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 оказалось недостаточным (иначе — перестало бы быть эффективным в этих условиях). Суть задачи была в том, чтобы спроектировать структуру, которая бы позволила приложению оставаться независимым, сохраняя работоспособность в файловой системе. Решение пришло от предшественника Mac OS X под названием NeXT, которая рассматривала приложение как *пакет* (bundle), содержащийся внутри *каталога* (directory). Концепция пакетов представляет подход группировки ресурсов приложения, исполняемых и других связанных файлов.

Если вы взглянете на любое приложения Mac, то обнаружите, что расширение app указывает не на файл, а на каталог. Это *программный каталог* приложения (program directory). Внутри он является организованной структурой, содержащей ресурсы, которые надо приложению для выполнения, информацию о приложении и исполняемые бинарные файлы приложения. iPhone SDK строит исполняемые бинарные файлы для вашей программы и размещает необходимые ему файлы в структуре этого программного каталога. Поэтому, чтобы создать цельное приложение, разработчику следует указать Xcode IDE, какие именно файлы необходимо установить. Приложения на iPhone выполняются в пределах *"песочницы"* (sandbox). "Песочница" — это ограниченная среда, запрещающая приложениям получать доступ к несанкционированным ресурсам. Одной из ее функций является запрет любых операций чтения и записи вне заданного домашнего каталога приложения. Все, что вашему приложению необходимо запускать, должно содержаться в пределах структуры такого каталога. Помимо этого, ваше приложение не будет знать, где именно оно установлено, поскольку при каждой установке ваше приложение получает уникальный идентификатор. Единственная возможность найти путь к вашему приложению — это воспользоваться функциями типа `NSHomeDirectory` и классами типа `NSBandle`, о которых вы узнаете в следующих главах.

Каждое приложение iPhone имеет собственный программный каталог, содержащий папки Library и Documents, а также каталог tmp для хранения временных файлов. Программный каталог для приложения iPhone гораздо менее структурирован, нежели для настольных приложений Mac. Все ресурсы приложения хранятся в корне программной папки .app. Далее приведен пример полного домашнего каталога одного приложения в том виде, который он мог бы иметь в файловой системе iPhone:

```
drwxr-xr-x mobile mobile Documents/
drwxr-xr-x mobile mobile Library/
drwxr-xr-x mobile mobile Preferences/
```

```
drwxr-xr-x mobile mobile MyApp.app/
    drw-r--r-- mobile mobile _CodeSignature
    -rw-r--r-- mobile mobile Default.png
    -rw-r--r-- mobile mobile icon.png
    -rw-r--r-- mobile mobile Info.plist
    -rwxr-xr-x mobile mobile MyApp
    -rw-r--r-- mobile mobile pie.png
    -rw-r--r-- mobile mobile PkgInfo
    -rw-r--r-- mobile mobile ResourceRules.plist
drwxr-xr-x mobile mobile tmp/
```

Приведенный далее список соответствует самому основному простейшему приложению iPhone под названием MyApp.

- Documents — специальная папка, в которой ваше приложение может хранить документы, создаваемые пользователем. Она не будет использоваться для хранения каких-либо других документов приложения.
- Library — папка, в которой ваше приложение может хранить настройки и другие ресурсы, создаваемые после установки. Внутри этой папки имеется другая папка — Preferences, которая будет содержать параметры предпочтений вашего приложения. В *главе 11* вы узнаете, как получить к ним доступ.
- MyApp.app — папка приложения, представляющая само приложение. Данный каталог будет содержать ваш исполняемый бинарный файл и все сопутствующие ресурсы, необходимые вашему приложению.
- _CodeSignature — каталог, содержащий подписи кода (code signatures) для каждого файла, задействованного вашим приложением. Это гарантирует то, что данное приложение не было изменено и имеет свой первоначальный вид. Чтобы выполняться на iPhone, все приложения должны быть подписаны.
- Default.png — файл изображения в формате PNG (Portable Network Graphics), содержащий заданный по умолчанию экран приветствия приложения. Когда пользователь запускает ваше 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 пикселов, перемещается на передний план и остается на экране до тех пор, пока приложение не закончит свою загрузку. Как правило, для фонового рисунка

приложения используют сплошной черный или белый цвета, логотип или фон, схожие с экраном приветствия, отображаемым приложением при своей инициализации.

- icon.png — изображение в формате PNG, содержащее значок приложения. Этот значок отображается на домашнем экране iPhone. Apple рекомендует использовать значки размером 57×57 пикселов. Сам файл может иметь любое имя, если оно указано в декларации Info.plist, рассмотренной далее. При отображении значки автоматически подсвечиваются, поэтому вам не нужно заботиться о том, чтобы сглаживать углы вашего значка и применять к нему световые эффекты.
- Info.plist — список свойств, содержащий информацию о приложении. Он содержит название исполняемого бинарного файла и идентификатор пакета, который считывается при запуске приложения. Далее в этой главе будет приведен пример списка свойств.
- MyApp — сам исполняемый бинарный файл, который вызывается при запуске приложения. Это именно то, что выдает на выходе Xcode при создании вашего приложения. При выполнении *Build and Go* Xcode автоматически помещает бинарный файл в папку приложения.
- pie.png — пример изображения ресурса, используемый данным простейшим приложением. Окружение iPhone предоставляет множество методов для получения ресурсов, поэтому вам не нужно получать к ним доступ напрямую с помощью пути. Такой подход соответствует попыткам Apple делать приложения независимыми. Xcode примет любые файлы, которые вы перетащите в папку Resources вашего проекта (на рабочем столе), и поместит их в программную папку приложения (на iPhone) после инсталляции проекта.
- PkgInfo — этот файл содержит восьмибайтовый дескриптор типа файла для данного приложения. Первые четыре байта должны считывать `APPL`, за которыми следует идентификатор подписи пакета. Об этом рассказывается в следующем разделе.

За кулисами Xcode

Прежде чем погрузиться в мир тех великолепных преимуществ, которые предоставляет Xcode, будет полезно разобраться с основными моментами сборки приложения. В старом мире взломов iPhone это означало ручную про-

катку вашего приложения. В Xcode это делается за вас. Вот что происходит за кулисами.

Если бы вы создавали приложение вручную, то составили бы скелет папки .app, предназначенной для его хранения. Скелет предоставляет всю информацию, необходимую iPhone для подтверждения существования вашего приложения как пакета, что позволит запускать его с домашнего экрана iPhone.

В этой книге представлено множество полнофункциональных примеров кода, скелеты которых были автоматически построены Xcode. После компиляции примера Xcode внутри каталога проекта build создает примерную структуру каталога и помещает его исполняемые файлы и ресурсы в папку приложения. Если бы вы делали это вручную, то создать такой каталог было бы весьма несложно:

```
$ mkdir MyExample.app
```

Затем Xcode копирует список свойств в папку приложения, чтобы описать само приложение и то, как его запустить. Файл Info.plist отображает эту информацию в формате XML и выглядит примерно так:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple//DTD PLIST 1.0//EN"
"http://www.apple.com
/DTDs/PropertyList-1.0.dtd">
<plist version="1.0">
<dict>
  <key>CFBundleDevelopmentRegion</key>
  <string>en</string>
  <key>CFBundleDisplayName</key>
  <string>${PRODUCT_NAME}</string>
  <key>CFBundleExecutable</key>
  <string>${EXECUTABLE_NAME}</string>
  <key>CFBundleIconFile</key>
  <string>icon.png</string>
  <key>CFBundleIdentifier</key>
  <string>com.yourcompany.${PRODUCT_NAME:identifier}</string>
  <key>CFBundleInfoDictionaryVersion</key>
  <string>6.0</string>
  <key>CFBundleName</key>
  <string>${PRODUCT_NAME}</string>
```

```
  <key>CFBundlePackageType</key>
  <string>APPL</string>
  <key>CFBundleResourceSpecification</key>
  <string>ResourceRules.plist</string>
  <key>CFBundleSignature</key>
  <string>????</string>
  <key>CFBundleVersion</key>
  <string>1.0</string>
  <key>LSRequiresIPhoneOS</key>
<true/>
</dict>
</plist>
```

Наиболее важными параметрами в приведенном примере являются `CFBundleDisplayName`, `CFBundleExecutable`, `CFBundleIconFile`, `CFBundleIdentifier` и `CFBundleName`. Свойство `CFBundleExecutable` имеет особую важность, поскольку задает имя исполняемого бинарного файла внутри папки. Это именно тот файл, который начинает исполняться при запуске вашего приложения, т. е. то, что выдает ваш компилятор. Как правило, это устанавливается Xcode автоматически, но вы можете переопределить такой порядок действий.

Домашний экран iPhone выполняется как приложение SpringBoard, которое аналогично приложению Finder, имеющемуся на рабочем столе Mac. Приложение SpringBoard, как и бóльшая часть уровня приложений iPhone, предпочитает ссылаться на приложения с использованием специального идентификатора, а не отображаемого имени приложения; например, com.yourcompany.tictactoe. Значение, присвоенное параметру `CFBundleIdentifier`, определяет уникальный идентификатор, который вы бы хотели дать вашему приложению. Всякий раз при запуске вашего приложения оно будет адресоваться 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 icon.png и Default.png, если они существуют, тоже должны быть скопированы. Если это будет упущено, то iPhone использует вместо них по умолчанию самые неприглядные изображения. Чтобы ваши приложения выглядели вполне профессионально, убедитесь, что при их публикации вы создали эти изображения и включили файлы этих изображений под такими именами.

Если данное приложение будет подписано Xcode, то этого будет вполне достаточно для его запуска. В следующем разделе вы узнаете, как установить на ваш рабочий стол iPhone SDK, который при построении приложения будет выполнять все эти шаги вместо вас, а также подписывать компоновку с использованием вашего ключа разработчика.

Вы приступите к компиляции примеров приложений в *главе 3*. В ближайших главах вы создадите множество примеров. Как правило, приводимые в этой книге примеры не требуют каких-либо дополнительных ресурсов, однако некоторые из них будут использовать ваше подключение к Интернету для загрузки примеров изображений или других файлов.

Примечание

Вы можете загрузить примеры кода, приводимые в этой книге, из хранилища книг в Интернете, расположенного по адресу: **http://www.oreilly.com/catalog/9780596154059**.

Установка iPhone SDK

Свою жизнь iPhone начал как полностью закрытая платформа. Еще до того, как Apple выпустила iPhone SDK, сообщество разработчиков открытого кода успешно взломало данное устройство и написало простейший компилятор для создания приложений. Позднее Apple наняла некоторых из разработчиков этого пакета инструментов с открытым кодом для проектирования iPhone SDK. В результате эти два пакета работают схожим образом: как кросс-компиляторы. *Кросс-компилятор* (cross-compiler) — это компилятор, который строит исполняемые файлы для другой архитектуры, нежели та, на которой он выполняется. В данном случае компилятор iPhone SDK выполняется на настольном компьютере с Mac OS X, но создает исполняемые файлы для используемой в iPhone архитектуры ARM.

Примечание

Пакет инструментов iPhone с открытым кодом может выполняться на множестве различных платформ, включая непосредственно iPhone, но не поддерживается Apple. Чтобы разрабатывать приложения, которые можно будет распространять посредством AppStore, вы должны использовать SDK. А это означает, что всем вашим разработчикам необходимо иметь Mac.

Все команды и пути к файлам, приводимые в этой книге, опираются на то, что установка iPhone SDK осуществлялась в соответствии с рекомендованными в этой главе процедурами. Apple периодически выпускает новые версии SDK, поэтому процесс его установки может иногда меняться. Более новые версии SDK можно найти на Web-узле Apple Developers Connection по адресу: **http://developer.apple.com/iphone**.

Что вам потребуется

iPhone SDK требует компьютер Mac на базе Intel, находящийся под управлением Mac OS X Leopard. Каждая версия SDK имеет собственные требования к версии операционной системы. Для SDK вам потребуется примерно 3,5 Гбайт дискового пространства, помимо дискового пространства для ваших собственных проектов.

Далее приведены требования, которые не являются обязательным, однако существенно облегчат разработку приложений для iPhone.

Ключ разработчика Apple

Ключи разработчика Apple (Apple developer keys) используются для подписи ваших приложений, чтобы вы могли устанавливать их на iPhone, для которых они разрабатываются. Apple выдает ключи разработчиков тем, кто принят в программу разработчиков Apple (Apple's developer program). Чтобы зарегистрироваться в этой программе, посетите Web-узел Apple Developers Connection по адресу: **http://developer.apple.com/iphone**. Для разработчиков Apple предлагает два пути: стандартный (standard track) и корпоративный (enterprise track). Стандартный путь предоставляет базовый ключ разработчика, позволяющий вам устанавливать приложения на вашем iPhone из Xcode. Как разработчик, вы сможете предлагать свои приложения для распространения через AppStore. Более дорогой корпоративный путь создан для корпораций, которые будут пользоваться разрабатываемыми приложениями только в рамках своих компаний или с выбранными партнерами, а не распространять их через AppStore. Этот путь включает дополнительные ключи инициализации для больших корпораций.

iPhone

Если вы хотите тестировать свои приложения непосредственно на самом iPhone, а не на платформе-симуляторе, то вам, естественно, понадобится

iPhone. Это крайне желательно. iPhone должен находиться под управлением той версии программного обеспечения, которая поддерживается вашей конкретной версией SDK.

Симулятор iPhone

Не имея ключа разработчика или iPhone, вам придется тестировать ваши приложения, используя симулятор iPhone. Симулятор iPhone — это целевая платформа, которую вы можете использовать для развертывания и тестирования на настольных системах приложений для iPhone. Симулятор предоставляет среду, подобную iPhone, с командами меню для симуляции блокировок, поворотов экрана и других базовых событий iPhone. Однако он сильно ограничен, поскольку на вашей настольной машине отсутствует необходимое аппаратное обеспечение для выполнения определенных задач. Используя симулятор, вы сможете получить только общее представление о том, как ваше приложение может функционировать, но вы упустите часть существенной функциональности:

- Core Location API не сможет предоставить вам ваши действительные координаты GPS, но сможет выдать вам данные-образец (sample) или общую информацию местоположения, доступную в вашей сети;
- приложению не будет доступно API акселерометра;
- в симуляторе доступны только некоторые жесты, например, пинч (pinch), и не поддерживаются жесты с использованием более чем двух пальцев, или беспорядочные множественные касания;
- приложение может не инициировать телефонные звонки;
- будут недоступны сети EDGE/3G, однако сетевые запросы будут использовать ваше подключение к Интернету в случае его доступности;
- не будут работать API камеры и API микрофона. Если ваше приложение поддерживает эту функциональность, то оно, возможно, будет выдавать исключения;
- доступными будут только определенные предварительно загружаемые приложения. Таковыми приложениями являются Contacts, Safari, Photos и Settings;
- вы не сможет видеть, истощили ли какие-либо части вашего приложения ресурсы процессора или память iPhone, поскольку ваша настольная машина обладает существенно бóльшими ресурсами для выполнения при-

ложений. До тестирования на iPhone трудно будет заметить, к примеру, медленную графическую обработку и другие проблемы подобного рода.

Загрузка и установка iPhone SDK

Загрузите iPhone SDK с Web-узла Apple Developers Connection, расположенного по адресу: **http://developer.apple.com/iphone**. Если у вас еще нет учетной записи (account) на этом Web-узле, то вам придется ее создать. Это делается бесплатно. Размер полного дистрибутива составляет примерно 2 Гбайт, поэтому загружать его лучше по высокоскоростному подключению к Интернету. Сам SDK является файлом образа диска (disk image file), который загружается в вашу заданную по умолчанию папку Downloads.

Чтобы смонтировать образ диска, дважды щелкните по нему. После этого вы увидите смонтированный том iPhone SDK. Он появится как на боковой панели вашего Finder, так и на рабочем столе. Откройте раздел — появится окно.

Рис. 1.1. Установщик iPhone SDK

Внутри этого окна вы увидите пакет iPhone SDK. Чтобы начать процесс установки, дважды щелкните по этому пакету. После того как вы согласитесь с различными лицензионными условиями, появится окно установки (рис. 1.1).

Убедитесь в том, что флажок **iPhone SDK** установлен, и нажмите кнопку **Continue**. Инсталлятор установит Xcode и iPhone SDK в папку /Developer вашего настольного компьютера.

Вот и все! Вы установили SDK для iPhone и теперь готовы приступить к компиляции приложений для iPhone. В следующем разделе мы обсудим, как пользоваться этим пакетом.

Инициализация iPhone

Если вы хотите установить ваши приложения на iPhone, то вам понадобится сертификат разработчика (developer certificate) и профиль мобильной инициализации (mobile provisioning profile) от Apple. Вы можете создать их с помощью портала iPhone Developer Program. Прежде чем создать профиль, вам придется заплатить взнос, чтобы присоединиться к одному из вариантов программы. Вы можете подписаться, зарегистрировавшись по адресу **http://developer.apple.com**. После того как вы будете приняты в выбранную программу разработчиков, вам будут предоставлены инструкции для получения доступа к порталу.

Чтобы подготовиться, выполните следующие несложные шаги. Поскольку интерфейс портала программы постоянно меняется, то пользуйтесь приведенными далее инструкциями только в качестве руководства и следуйте непосредственным инструкциям на портале:

1. Войдите на портал программы. Первое, что нужно сделать, — это создать сертификат разработчика. Xcode использует этот сертификат для подписания ваших приложений. Чтобы добавить новый сертификат, щелкните по вкладке **Certificates** и последуйте инструкциям. После создания сертификата загрузите его и WWDR Intermediate Certificate согласно приводимой на портале информации. WWDR Intermediate Certificate — это ключ Apple, который вам также потребуется. Чтобы добавить сертификаты в вашу связку ключей (keychain), после их загрузки дважды щелкните по каждому из них.
2. Теперь зарегистрируйте ваш iPhone на портале, щелкнув по вкладке **Device**. Регистрация вашего iPhone необходима, поскольку устанавливать

свои тестовые приложения вы можете только на зарегистрированных устройствах. При регистрации вам потребуется указать уникальный идентификатор устройства (device ID) вашего iPhone, который вы можете получить из органайзера устройств (device organizer) Xcode. Для этого запустите Xcode, затем перейдите в меню **Windows** и выберите команду **Organizer**. В результате появится диалоговое окно (рис. 1.2), в котором будет приведен список устройств. Подключите ваш iPhone, и вам будет предложено использовать ваше устройство для разработки приложений. Щелкните по данному устройству в левой панели органайзера. Появится информационное окно, содержащее идентификатор устройства вашего iPhone. Используйте его для регистрации вашего iPhone на портале программы.

Рис. 1.2. Окно **Organizer**

3. Далее, чтобы создать идентификатор пакета приложения, щелкните по вкладке **App IDs**. Этот идентификатор определяет приложение (или группу приложений), которое вы будете разрабатывать. В нем могут использо-

ваться групповые символы (wildcard), что позволит вам устанавливать любые приложения, используя заданный префикс пакета. Чтобы запустить приводимые в этой книге примеры на вашем iPhone, создайте групповой символ, используя * в качестве идентификатора пакета com.yourcompany.*. Само приложение вы можете назвать Examples или так, как вам захочется.

4. Затем создайте профиль инициализации (provisioning profile) для вашего мобильного устройства. Профиль инициализации позволит вам устанавливать приложения, скомпилированные с помощью только что созданного вами на вашем iPhone идентификатора. Выберите идентификатор приложения для ассоциирования его с данным профилем, а также сертификаты разработчика и устройства для использования с данным профилем. После создания профиля загрузите его в вашей настольной системе.
5. Чтобы добавить профиль инициализации на ваш iPhone, в органайзере щелкните по значку "плюс" (+), расположенному под полем **Provisioning**, и перейдите к вашему сертификату инициализации. После завершения установки вы сможете на данное устройство устанавливать приложения из Xcode.

Примечание

Помимо всего, в окне органайзера вы можете просматривать журналы консоли и аварийных ситуаций вашего устройства, а также делать снимки экрана (screenshot).

Построение и установка приложений

Следующим шагом после установки iPhone SDK является изучение того, как им пользоваться. iPhone SDK работает как компонент Xcode IDE. Чтобы запустить Xcode, перейдите к недавно созданному каталогу /Developer на вашем жестком диске. Откройте окно Finder, а затем на боковой панели щелкните по значку диска. Теперь дважды щелкните по папке Developer, за которой следует папка Applications. Чтобы облегчить запуск Xcode, перетащите приложение Xcode на вашу док-панель (dock).

Примечание

Папка /Developer/Applications отделена от заданной по умолчанию папки Mac OS /Applications. Вы можете перетащить эту папку на док-панель как стек (stack), чтобы сделать легко досягаемыми все приложения для разработки, или же перетащить ее на боковую панель, чтобы вызывать их с по-

мощью Finder. По мере разработки программного обеспечения вы будете часто обращаться к этой папке, поэтому вам в любом случае захочется облегчить к ней доступ.

После запуска Xcode в меню **File** выберите пункт **New Project**. Появится диалоговое окно, предлагающее вам указать шаблон для вашего нового проекта. Под заголовком **iPhone OS** щелкните по категории **Applications**. На выбор вам будет представлено несколько различных типов шаблонов приложений (рис. 1.3).

Рис. 1.3. Шаблоны приложений для iPhone

Модель — Представление — Контроллер

Процесс разработки программного обеспечения на iPhone следует парадигме "Модель — Представление — Контроллер" (Model — View — Controller, MVP).

Суть такой архитектуры состоит в том, чтобы отделить бизнес-логику, например, данные вашего приложения и управляющие ими правила, от компонентов пользовательского интерфейса (UI), отображаемых конечному пользователю. Для реализации архитектуры MVP необходимы три ключевых компонента. *Модель* (model) представляет данные и бизнес-логику вашего приложения. *Представление* (view) реализует элементы пользовательского интерфейса, которые отображают данные пользователю и позволяют выполнять над ними необходимые действия. *Контроллер* (controller) обеспечивает взаимодействие между элементами пользовательского интерфейса и данными, например, ответные реакции на жесты со множественными касаниями, события взаимодействия и переходы между различными частями логики.

Как вы увидите далее, все эти принципы повлияли на названия, данные многим классам iPhone. Зачастую контроллеры также инкапсулируют представление, облегчая тем самым управление представлением и избавляя от написания большого количества кода для подключения к нему.

Шаблоны приложений

Для реализации архитектуры MVC в ваших приложениях Xcode предоставляет несколько конструкций. Наиболее распространенными являются следующие шаблоны.

- **View-Based Application** (Приложение на базе представления). Этот шаблон должны применять приложения, которые используют только одно представление. Простой контроллер представлений управляет основным представлением приложения, используя для раскладки шаблон построителя интерфейса (interface-builder), но мы покажем вам, как при желании его удалить и создать собственный. Этот шаблон должны применять простые приложения, не использующие какую-либо навигацию. Если вашему приложению требуется навигация по различным представлениям, то рассмотрите вариант использования шаблона, основанного на навигации (navigation-based).
- **Navigation-Based Application** (Навигационное приложение). Шаблон, основанный на навигации, идеально подходит для приложений, обладающих несколькими различными представлениями и нуждающихся в средствах для навигации между ними. Если вы можете представить в своем приложении присутствие кнопки **Back** (Назад), то, скорее всего, вам необходим именно этот шаблон. Контроллер навигации берет на себя всю внутреннюю работу по настройке кнопок навигации и переходам между

"стеками" представлений. Для отображения информации данный шаблон предоставляет основной контроллер навигации и контроллер корневого (base-tier) представления.

- **Utility Application** (Служебное приложение). Идеально подходит для приложений типа виджетов (widget-type), в которых приложение имеет главное представление, которое вы можете "перевернуть" ("flip" around), как виджет в Leopard. Возможно, вы знакомы с этим еще по Konfabulator (предшественнику Dashboard от Apple, разработанному сторонней компанией). Приложения Weather (Погода) и Stocks (Акции) на iPhone являются хорошими примерами служебных приложений. Помимо всего прочего, данный шаблон включает в себя информационную кнопку, нажатие которой переворачивает представление, чтобы отобразить обратную сторону приложения, которая обычно используется для внесения изменений в параметры или в отображаемую информацию.
- **OpenGL ES Application** (Приложение OpenGL ES). Используйте этот шаблон, если вы разрабатываете 3D-игры или графические приложения. Он создает отображение, предназначенное для визуализации сцен GL, и предоставляет простой таймер для их анимации. В данной книге не рассматривается программирование с использованием OpenGL.
- **Tab Bar Application** (Приложение с панелью вкладок). Данный шаблон предусматривает специальный контроллер, отображающий вдоль нижней границы экрана панель кнопок. Этот шаблон идеально подходит для приложений iPod или Mobile Phone, где панель вкладок внизу предоставляет набор ярлыков для быстрого доступа к ключевой функциональности приложения.
- **Window-Based Application** (Оконное приложение). Если ни один из описанных выше пяти шаблонов вам не подходит, то можете воспользоваться этим весьма простым шаблоном, который предоставляет простое приложение с окном. Это самый минимум, необходимый вам для начала создания вашего собственного приложения.

Содержимое проекта Xcode

После создания нового проекта его содержимое будет располагаться в отдельном окне, наподобие изображенного на рис. 1.4. Этот проект содержит в себе источники (sources), библиотеки (frameworks) и ресурсы (resources) для приложения.

Рис. 1.4. Только что созданный проект iPhone

Приведенные далее группы могут помочь упорядочить ваш проект.

- **Classes** (Классы). Файлы, содержащие классы Objective-C, которые использует ваше приложение. Сюда включаются объект-делегат приложения, контроллеры представлений и другие создаваемые вами объекты. Как только вы добавите в ваше приложение новые файлы классов, они появятся в этой группе.
- **Other Sources** (Другие источники). Другие источники, скомпилированные в ваше приложение. По умолчанию сюда включаются прекомпилированные заголовочные файлы и функция `main` вашего приложения, которая создает экземпляр объекта приложения Cocoa. Если ваше приложение включает в себя дополнительные функции C или классы C++, то добавьте сюда соответствующие файлы.
- **Resources** (Ресурсы). Ресурсы приложения, которые не компилируются вместе с исходным кодом вашего приложения, но копируются в программную папку при построении приложения. Сюда входят изображения или звуковые файлы, игровые уровни и другие важные файлы.

- **Frameworks** (Библиотеки). Библиотеки, которые подключает ваше приложение. Это разделяемые библиотеки (shared libraries), которые подключаются во время компоновки для обеспечения дополнительной функциональности. Например, 3D-игра подключает библиотеку OpenGL, которая содержит процедуры для визуализации 3D-графики. Приложения, активно использующие звуковые эффекты, скорее всего, будут пользоваться преимуществами, даваемыми библиотеками Audio Toolbox и AVFoundation, которые содержат процедуры для воспроизведения и микширования звуков различных типов.
- **Products** (Продукты). Целевая компоновка для вашего приложения.

Прототипы

В этой книге в конце большинства разделов будет даваться соответствующий список прототипов. Прототипы (prototypes) — это заголовочные файлы, содержащие список поддерживаемых методов и свойств, которые вы, как разработчик, можете использовать. Хотя эта книга и описывает подавляющее большинство существующих методов и свойств, тем не менее, самостоятельное изучение прототипов может открыть вам дополнительные, а иногда скрытые интерфейсы, которые, возможно, были недокументированны или добавлены после выхода этой книги. Они также покажут вам, какие аргументы принимает тот или иной метод, и какого типа данные он будет возвращать.

Заголовочные файлы хранятся в подпапке Headers папки библиотек. Поскольку существуют различные версии SDK, то путь, по которому могут быть найдены эти файлы, может несколько отличаться. Стандартный формат для пути к этим прототипам следующий: /Developer/Platforms/*PLATFORM*/Developer/SDKs/*VERSION*/System/Library/Frameworks/.

Полный путь использует две переменные: *PLATFORM* и *VERSION*. В SDK существуют две основные платформы: iPhoneOS.platform и iPhoneSimulator.platform. Первую вы используете при создании приложения для устройств iPhone или iPod Touch, а платформу симулятора — при создании приложения для симулятора iPhone. Каждая платформа содержит собственный набор библиотек, разделяемых библиотек и прототипов.

Переменная *VERSION* обозначает версию iPhone SDK для данной платформы. В начале версии идет название платформы, а в конце — расширение sdk.

Полный путь к платформе iPhone для SDK версии 2.2 имеет следующий вид: /Developer/Platforms/iPhoneOS.platform/Developer/SDKs/iPhoneOS2.2.sdk/System/Library/Frameworks/.

С целью упростить поиск этого каталога добавьте в ваш profile-сценарий приведенные далее строки, чтобы переменная окружения `SDK` задавалась всякий раз при создании новой оболочки:

```
export PLATFORM=iPhoneOS
export VERSION=2.2
export SDK=/Developer/Platforms/${PLATFORM}.platform/Developer/SDKs\
/${PLATFORM}${VERSION}.sdk/System/Library/Frameworks
```

Затем вы сможете изменить каталог с помощью переменной окружения `SDK`:

```
$ cd ${SDK}
```

Внутри папки по этому пути вы найдете отдельные библиотеки, имеющиеся в SDK, каждая из которых имеет каталог Headers, содержащий прототипы для данной библиотеки. Обязательно ознакомьтесь с ними, когда вам будет предложено, поскольку они содержат огромное количество информации о том, что могут использовать разработчики. Главное правило состоит в следующем: если в заголовках SDK вы находите класс, свойство или метод, он должен быть санкционирован к использованию в вашем приложении.

> ***Примечание***
>
> Хотя заголовки библиотеки проинформируют вас о том, какой API вы можете использовать, они совсем не обязаны сообщать вам, *как* его использовать. Apple поддерживает целый ряд руководств по созданию интерфейсов с пользователем, а также другие политики, регулирующие правила создания программного обеспечения AppStore. Вам необходимо будет убедиться в том, что ваше приложение не нарушает никаких нетехнических ограничений Apple, например, дублирования функциональности существующих предварительно поставляемых приложений. Сверьтесь с последней версией вашего соглашения SDK и с другими соответствующими документами от Apple, чтобы узнать о подобных ограничениях в проектировании.

Добавление библиотек

Все классы и методы, необходимые для предоставления какого-либо определенного заданного типа функциональности, собраны в библиотеку (framework). Например, библиотека Core Location предоставляет всю функциональность по глобальному местоопределению, библиотека UI Kit — всю функциональность по реализации пользовательских интерфейсов. Чтобы воспользоваться предоставляемой какой-либо библиотечной функциональностью, вам необ-

ходимо *подключиться* (link) к этой библиотеке. Такая модель логически отделяет различные части операционной системы iPhone для разработчиков и позволяет подключаться только к требуемым компонентам.

Примечание

Когда вы подключаетесь к какой-либо библиотеке, все ее классы, функции и символы становятся доступными вашему приложению, как если бы вы сами их написали. *Статическое* связывание (static linking) компилирует объекты напрямую в ваше приложение. *Динамическое* связывание (dynamic linking) загружает объекты во время исполнения. Добавление библиотеки приводимым далее способом использует динамический метод связывания.

В этой книге вам будет предложено добавить одну или две библиотеки в ваш пример, чтобы обеспечить поддержку функциональности определенного рода. Для этого в вашем проекте Xcode щелкните правой кнопкой мыши по папке Frameworks и в появившемся контекстном меню выберите команду **Add | Existing Frameworks** (рис. 1.5). Перейдите к каталогу iPhone SDK Frameworks и выберите папку соответствующей библиотеки. После нажатия кнопки **Add** новая библиотека появится в папке Frameworks вашего проекта. Таким образом, библиотека будет подключена к вашему приложению.

Примечание

Может потребоваться перейти к папке Frameworks внутри вашего SDK. Чтобы найти прототипы, используйте только что рассмотренные файловые пути к ним.

Рис. 1.5. Добавление существующей библиотеки в Xcode

Установка активного SDK

Xcode позволяет вам создавать приложения как для физически существующего устройства iPhone, так и для интегрированного симулятора iPhone. Для какой-либо конкретной версии встроенного программного обеспечения iPhone можно использовать различные версии SDK. Чтобы переключаться между сборкой для физического устройства и сборкой для симулятора, или для смены версии SDK, в Xcode щелкните по меню **Project**. Прокрутите до пункта меню **Set Active SDK** и выберите тот SDK, для которого вы хотите осуществить сборку. Вы также можете выбрать, как осуществлять сборку: с поддержкой отладки (debugging) или без нее. Для этого выберите соответствующий подпункт в пункте меню **Set Active Build Configuration**. Сборка для конфигурации Debug позволит вам распознавать потенциальные проблемы путем трассировки выполнения вашего приложения.

Построение приложения

Существуют два способа построения приложения с помощью Xcode: GUI и командная строка. Конечно, самый простой путь — это щелкнуть по кнопке **Build**, расположенной вверху окна проекта, или нажать кнопку **Build and Run**, чтобы построить и запустить приложение. Это приведет к вызову компилятора и выводу результатов сборки в строку состояния. Если сборка не удастся или в ее процессе возникнут предупреждения (warnings), то вы сможете получить дополнительную информацию, щелкнув по соответствующему значку в строке состояния.

Если у вас есть опыт работы в UNIX или вы просто любите трудности, то вам, возможно, будет более комфортно осуществлять сборку приложений из командной строки, особенно если вы вместо Xcode IDE используете текстовый редактор командной строки. Чтобы собрать приложение из командной строки, воспользуйтесь командой `xcodebuild`:

```
$ xcodebuild -target Project_Name
```

Установка приложения

После сборки приложения вы можете установить его на ваш iPhone прямо из Xcode. Как говорилось ранее, это потребует действующего ключа разработчика от Apple. После того как приложение будет установлено, оно сразу же появится на домашнем экране iPhone, а затем будет запущено отладчиком.

Чтобы установить из Xcode, воспользуйтесь кнопкой **Build and Go** на панели инструментов. Таким образом, будут перекомпилированы все изменения, сделанные с последней сборки, а новая сборка будет установлена на iPhone. Если у вас нет ключа разработчика, и вы собираете приложение для платформы симулятора iPhone, то это приложение будет установлено на симулятор iPhone, а затем запущено.

Чтобы установить приложение из командной строки, воспользуйтесь командой `xcodebuild` с параметром сборки `install`:

```
$ xcodebuild install -target Project_Name
```

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 среды iPhone написана на Objective-C, но поскольку этот язык проектировался так, чтобы обеспечить совместимость с языком C, то вы вполне свободно можете использовать в вашем приложении C и C++. Objective-C в основном применяется в Mac OS X и GNUStep (бесплатная среда 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 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 данной книги в качестве руководства.

Обмен сообщениями

Однако бóльшую часть всего этого можно отнести к семантике, по крайней мере, то, с чем сталкивается разработчик. Работа программы вашего приложения нисколько не похожа на программу на C или C++: когда вы вызываете

метод, ваша программа, прежде чем продолжить, ждет ответа, позволяя вам присвоить возвращаемое значение или вызвать методы как часть условных операторов.

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 или проводить тестирование того, отвечает ли какой-либо объект на конкретное сообщение. 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. Например, вы можете скомпилировать код, который будет посылать объекту следующее сообщение:

```
[ myObject heyWhatsUpHowYouDoin ];
```

Во время исполнения данное приложение выдаст исключение, если только для ответа на это сообщение не существует метода `heyWhatsUpHowYouDoin`. Конечно, хорошо, если такой метод для данного объекта существует, но недокументирован в прототипах класса. Мы рассмотрим несколько примеров недокументированных методов, подобных этому, в *главе 3*.

При наличии объекта `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 плоского пространства имен:

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

С сообщениями также могут передаваться аргументы, если 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 объявлять несколько методов с одинаковыми именами *и* типами данных — полиморфизм на стероидах (polymorphism on steroids):

```
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 функциональностей Objective-C. Это распространяется и на использование 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 реализацию (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 1.1, являющемся файлом с именем MyWidget.h.

Листинг 1.1. 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.

Тип данных *id*

В только что показанном примере возвращаемое значение метода `alloc` имеет тип `id`. Этот тип данных является общим типом (generic type), используемым для ссылки на любой произвольный объект. Воспринимайте тип данных `id` как указатель на объект неизвестного типа (void pointer) в Objective-C. Где бы он ни был определен, может быть использован любой объект. В этой книге вы будете встречать этот тип данных особенно в случаях, когда делегат присваивается объекту для получения специальных уведомлений о событиях.

Импорт

Директива препроцессора `#import` замещает традиционную директиву `#include` (хотя `#include` все равно может использоваться). Единственное преимущество применения `#import` состоит в том, что она обладает 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 применяемых в коде C:

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

Объявление интерфейсов

Интерфейс для какого-либо класса объявляется оператором `@interface` с последующим указанием имени интерфейса и базового класса (если таковой есть), от которого он был порожден. В конце объявления вашего класса блок заканчивается оператором `@end`. Внутри этого блока вы объявляете все переменные класса и методы для данного класса.

Методы

Методы должны объявляться вне фигурных скобок. Плюс (+) определяет метод как статический, а минус (–) определяет метод методом экземпляра. Для вызова статического метода не требуется конкретный объект; они являются методами, представляющими весь класс целиком, — например, "the Widget class". Методы экземпляра вызываются для определенного экземпляра класса — например, "my Widget" и "his Widget".

Метод `alloc` является хорошим примером статического метода. Он отвечает за размещение и возврат нового экземпляра для заданного класса. В вашем коде вы будете вызывать этот метод всякий раз, когда будете создавать новый объект, ссылаясь на данный класс напрямую, например, `myWidget`. Методы экземпляра, являющиеся специфическими для экземпляра класса `MyWidget`, например, `setSpeed` и `powerOn`, должны вызываться путем ссылки на именованный объект. Многие классы Cocoa предоставляют статические методы и методы экземпляров для инициализации объектов.

Для каждого объявляемого аргумента метода указывается тип данных, имя локальной переменной и необязательное внешнее имя переменной. Примерами внешних имен переменных в листинге 1.2 являются `withMass` и `WithGyroscope`. Вызывающий метод (notifier), который вызывает данный метод, использует внешние (public) имена аргументов, но внутри этого метода на аргументы ссылаются, используя их локальные имена. Таким образом, метод `setSpeed` использует локальную переменную `_mass` для получения значения, передаваемого как `withMass`.

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

Реализация

Суффиксом для файла исходного кода на Objective-C является .m. Реализация скелета простейшего класса из последнего раздела может быть такой, как показано в листинге 1.2, названном MyWidget.m.

Листинг 1.2. Пример реализации (MyWidget.m)

```
#import "MyWidget.h"

@implementation MyWidget

+ (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 единый блок кода, то реализация начинается с оператора `@implementation` и заканчивается оператором `@end`.

Весьма распространенной практикой в C++ является использование префикса для переменных экземпляра, чтобы внешние методы могли принимать действительные имена переменных. Это облегчает повторное использование чужого кода, поскольку вы можете догадаться о назначении переменной по ее имени. Так как Objective-C разрешает использование внутренних и внешних имен переменных, то метод может предоставлять разработчику осмысленное имя, в то время как внутри использовать собственное имя. Это истинное имя затем может быть использовано в качестве параметра данного метода, а имя локальной переменной метода иметь в качестве префикса символ подчеркивания, например, `_speed`.

Рассмотрим только что показанный пример метода `setSpeed`. При вызове используется его внешнее имя аргумента:

```
[ myWidget setSpeed: 1.0 withMass: 2.0 ];
```

Внутри блока кода данного метода используется имя внутренней переменной `_mass`:

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

Свойства

Objective-C 2.0 вводит понятие свойств (properties). Свойства — это посредники между переменными экземпляра и методами; они придают удобства с точки зрения синтаксиса, позволяя разработчику адресовать переменные напрямую, а не посредством отдельного метода считывания (getter) и отдельного метода задания (setter). Свойства похожи на глобальные переменные, однако разрешают задавать способы доступа к ним. Кроме того, свойства вызывают методы, которые могут быть подменены, когда это свойство считывается или записывается.

Чтобы работать с переменной экземпляра в более ранних версиях Objective-C, как правило, вам приходилось писать два метода — один, чтобы считывать переменную (getter), а другой, чтобы присваивать ей значение (setter):

```
BOOL myVariable = [ myClass variable ]; /* getter */
[ myClass.setVariable ] = YES;          /* setter */
```

Свойства позволяют разработчику поменять синтаксис, чтобы обращаться к обеим операциям по имени свойства. Тем самым код становится более похожим на код на языке C:

```
BOOL myVariable = myClass.variable;
myClass.variable = YES;
```

Вы можете описать свойства с целым рядом различных семантик хранения, включая `assign`, `retain` и `copy`. Кроме того, вы можете задать свойства как `readonly`. Свойство для переменной `size` в описании класса описывается следующим образом:

```
@interface MyClass : BaseClass
{
  int size;
}
@property(copy) int size;
```

Чтобы реализовать данное свойство, в реализации данного класса используйте оператор `@synthesize`:

```
@implementation MyClass
@synthesize size;
...
@end
```

Если используется оператор `@synthesize`, то метод считывания (getter) и метод задания (setter), если они не существуют в вашем коде, автоматически генерируются внутри. Эти методы явно вызываются всякий раз при осуществлении доступа к данному свойству:

```
myClass.size = 3;
```

Также вы можете описать собственные метод считывания (getter) и метод задания (setter), не прибегая к использованию оператора `@synthesize`:

```
-(int)myGetter;
-(void)setSize:(int)size;

@property(copy,getter=myGetter) int size;
```

Очень многие методы, которые вы встречали в предыдущих версиях кода на Objective-C, были заменены свойствами в iPhone SDK.

Протоколы

Протокол (protocol) — это набор методов, которые объект согласен реализовать для того, чтобы взаимодействовать с другим объектом. В этой книге представлено много протоколов, известных как *протоколы-делегаты* (delegate protocols). Протоколы-делегаты — это протоколы, которые уведомляют объект о событиях, произошедших в другом объекте. Чтобы реализовать протокол, ваше объявление интерфейса просто заявляет, что поддерживает заданный протокол. Тогда другие классы, требующие определенного протокола, могут ожидать, что ваш класс будет отвечать на методы, использованные в данном протоколе.

Класс может реализовывать произвольное количество протоколов. Если ваша реализация будет неполной, то вы получите предупреждение компилятора.

В качестве примера предположим, что протокол `BaseWidgetDelegate` является протоколом, который вы разработали для предупреждения объекта о событиях, происходящих внутри заданного виджета, например, событие включения питания или событие изменения параметров его питания.

Для описания методов, используемых в протоколе, воспользуйтесь оператором `protocol`. С помощью оператора `required` вы можете задать методы, обязательные для реализации протокола:

```
@protocol BaseWidgetDelegate

- (void)baseWidgetDidChangePowerSettings:(float)newPowerLevel;
```

```
@required
- (void)baseWidgetDidGetTurnedOn:(id)widget; /* Это обязательный метод */

@end
```

Класс, нуждающийся в отправке уведомлений такого рода, может определить переменную, содержащую указатель на объект для получения уведомлений:

```
@interface MyWidget : BaseWidget {
  id delegate;
}
```

Можно описать свойство для делегата и потребовать, чтобы назначенный ему объект реализовывал протокол `BaseWidgetDelegate`:

```
@property(assign) id<BaseWidgetDelegate> delegate;
```

Класс, желающий получать уведомления, обязан теперь реализовать протокол `BaseWidgetDelegate`, чтобы стать назначенным в качестве делегата виджета. Приведенный далее пример объявляет класс `BaseWidgetManager` и задает протокол для реализации внутри класса:

```
@protocol BaseWidgetDelegate;
@interface WidgetManager : NSObject <BaseWidgetDelegate>
...
@end
```

Единственное, что осталось реализовать, — это сами методы протокола в реализации данного класса.

После реализации код вашего приложения может назначить класс менеджера виджета свойству виджета `delegate` без генерирования предупреждения компилятора, поскольку он реализует протокол `BaseWidgetDelegate`:

```
myWidget.delegate = myWidgetManager;
```

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 трудно определить, какие изменения безопасны. Категории предоставляют механизм для добавления функциональности меньшим классам, не затрагивая остальной код.

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 исходную версию. С точки зрения разработки это существенно облегчает процесс усовершенствования класса, написанного другим разработчиком. Во время исполнения те части кода, которые используют категорию, будут видеть новую версию класса, а код, использующий базовый класс напрямую, будет видеть только исходную версию.

Возможно, вы подумали о наследовании, как о возможном равноценном решении. 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 она начинает иначе функционировать. Вы можете даже полностью вытащить какие-либо компоненты и заменить их новыми. 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 и папье-маше прикрепляются к автомобилю с внешней стороны, делая его неузнаваемым. В контексте парада автомобиль для зрителя выступает в роли совершенно иного предмета, но для механика он все тот же старый автомобиль, которым вы управляете много лет.

В качестве примера рассмотрим ситуацию, когда завод по производству виджетов выпустил новый тип виджетов, которые могут летать в космическом пространстве, но руководство завода беспокоится о том, что внесение изменений в базовый класс может разрушить существующее приложение. Путем создания категории разработчики гарантируют, что приложения, использующие базовый класс `MyWidget`, продолжат видеть исходный класс, в то время как новые космические приложения вместо него будут использовать категорию. Приведенный далее пример создает категорию `MySpaceWidget`

поверх существующего базового класса `MyWidget`. Поскольку нам потребуется уничтожение различных вещей в космическом пространстве, то мы добавили метод `selfDestruct`. Кроме того, данная категория замещает метод `powerOn` собственным. Сравните использование здесь круглых скобок для `MySpaceWidget`, содержащего класс, с использованием двоеточия в листинге 1.1 для обеспечения наследования:

```
#import "MyWidget.h"

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

В листинге 1.3 приведен весь исходный файл, реализующий категорию.

Листинг 1.3. 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
```

Подстановка

В Objective-C вместо одного из его суперклассов можно *подставить* (pose) какой-либо подкласс, фактически заместив 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. Для подставляемого класса нельзя объявлять никаких новых переменных, хотя он и может подменять или замещать существующие методы. Подстановка схожа с категориями в том, что она позволяет разработчику дополнять существующий класс непосредственно во время исполнения.

В предыдущих примерах были созданы классы механических виджетов (widgets). Теперь после создания всех этих виджетов в некоторый момент появилась необходимость в проведении дополнительной диагностики. Вместо того чтобы переделывать функции в исходном классе, вы решили создать новый подкласс `MyDiagnosicsWidget` (листинги 1.4 и 1.5).

Листинг 1.4. Пример интерфейса подстановки (MyDiagnosticsWidget.h)

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

@interface MyDiagnosticsWidget : MyWidget
{

}

- (void)debug;
@end
```

Листинг 1.5. Пример реализации подстановки (MyDiagnosticsWidget.m)

```
#import "MyDiagnosticsWidget.h"

@implementation MyDiagnosticsWidget

- (void)debug {
    /* Генерируем отладочную информацию */
}
@end
```

Вместо того чтобы для использования этого класса изменять весь существующий код, автономный класс можно просто подставить вместо класса виджета. Метод `class_poseAs` вызывается из основной программы или другого метода высокого уровня, чтобы привести к следующему поведению:

```
MyDiagnosticsWidget *myDiagWidget = [ MyDiagnosticsWidget alloc ];
    MyWidget *myWidget = [ MyWidget alloc ];
    class_poseAs(myDiagWidget, myWidget);
```

С этого момента все остальные методы, замененные вами в подставляемом классе, будут подставляться вместо методов исходного базового класса.

Дополнительные источники

Во многих примерах, приводимых в этой книге, вы будете встречать ссылки на объекты, используемые в среде Cocoa от Apple. Бóльшая часть этих объектов начинается с префикса `NS`, как например, `NSError` или `NSString`. Среда Cocoa предоставляет целый ряд таких стандартных объектов для оперирования массивами, строками и другими подобными объектами; многие из них доступны как в среде iPhone, так и в настольной среде Mac OS X. Мы покажем вам, как работать с некоторыми из этих объектов в приводимых примерах, однако только для документирования классов Cocoa написаны целые тома. Если вы встретите незнакомый вам класс, то сможете найти полное его описание в доступной в Интернете документации от Apple. На Web-узле Apple Developer Connection хранится справочник по Cocoa, доступный по адресу: **http://developer.apple.com/reference/Cocoa/**. В этом справочнике вы можете найти какую-либо определенную тему или же в окне поиска ввести название интересующего класса, чтобы сразу же перейти к описанию данного класса.

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

- James Duncan Davidson и Apple Computer Inc., "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 2

Interface Builder: Xcode GUI для графических пользовательских интерфейсов

Одним из самых затратных по времени шагов при написании приложения может быть проектирование пользовательского интерфейса. iPhone обладает пользовательским интерфейсом, который невозможно перепутать ни с одним другим мобильным устройством, — полнофункциональный, с высоким разрешением, чтобы обеспечивать навигацию без применения стилуса (т. е. способом, удобным для пальцев). Хакеры старой школы, которые писали пользовательский интерфейс вручную, скажут вам, что им для этого требовалось огромное количество кода и гораздо больше времени, нежели им хотелось бы. Отличной новостью является то, что Apple прошла большой путь, чтобы упростить для разработчиков SDK реализацию стандартного набора элементов пользовательского интерфейса. iPhone SDK включает в себя служебную программу Interface Builder, которая предоставляет разработчику среду WYSIWYG с применением техники drag-and-drop для создания представлений и размещения элементов управления на экране без особого труда. Пользователь может сохранить расположение элементов на экране как файл шаблона, который ваше приложение может прочитать и синтезировать (synthesize) как объект в вашем приложении. Это позволяет вам создавать пользовательский интерфейс за считанные минуты и вносить в него последующие корректировки с небольшими изменениями кода, а то и вовсе без них. Разработчики настольных систем, возможно, помнят такой подход при работе с ресурсными потоками (resource forks) в прошлом.

Приложение Interface Builder можно найти в папке приложений /Developer на вашем настольном компьютере. Если вы последовали нашему совету из *главы 1*, то уже перетащили эту папку на вашу док-панель (dock) и можете запустить это приложение, открыв стек док-панели и выбрав приложение Interface Builder. Если вы не прислушались к нашему мудрому совету, то вам придется обратиться в Finder к вашему жесткому диску и запустить Interface Builder из папки /Developer/Applications.

Поскольку использование Interface Builder не мешает создавать собственные объекты пользовательского интерфейса, то многие разработчики предпочитают полностью отказаться от Interface Builder и описывать все компоненты пользовательского интерфейса внутри собственного кода. О том, как это делать, вы узнаете в конце данной главы.

Примечание

Синтезирование (synthesizing) — термин, используемый Apple для описания динамического создания взаимосвязи между переменной и объектом, с которым она связана. Когда какой-либо класс синтезирует объект из шаблона Interface Builder, данный объект присваивается переменной внутри вашего класса. Контроллеры представлений, окна и объекты Interface Builder других типов могут быть синтезированы, позволяя вам получать к ним доступ как к переменным без необходимости создавать их напрямую в вашем приложении.

Окна, представления и контроллеры представлений

Окна и представления являются базовыми классами для создания пользовательского интерфейса любого типа. Окно (window) представляет собой геометрическое пространство на экране, а представление (view) выступает в роли холста для других объектов. Все небольшие компоненты пользовательского интерфейса, например, кнопки и текстовые поля, привязаны к представлению, а уже это представление прикреплено к какому-либо окну. Окно можно рассматривать как рамку картины, а представление — как само полотно. Окно содержит только классы представлений, а представления могут содержать элементы управления, изображения и даже подчиненные представления, например, таблицы и выборщики (pickers). О различных компонентах пользовательского интерфейса вы узнаете в *главах 3* и *10*.

Контроллер представлений (view controller) — это объект представления специального типа, который управляет тем, как данное представление отображается на экране. Контроллеры представлений могут инкапсулировать один или более объектов представления и иметь возможность обрабатывать повороты экрана и переходы к другим представлениям, когда пользователь нажимает какую-либо кнопку или переворачивает страницу. Контроллер представлений разработан как посредник между отображаемым им представлением и вашими данными и бизнес-логикой.

Для предоставления различных способов отображения пользователю ваших данных существуют различные типы контроллеров представлений. Приложения iPod используют набор вкладок вдоль нижней части экрана, позволяющих вам переключаться между списками исполнителей и песен, видеороликами и представлениями с другой информацией. Сама по себе панель вкладок имеется в одном типе класса контроллеров представлений (контроллер панели вкладок — tab bar controller), который всегда виден на экране независимо от того, какую страницу просматривает пользователь в данный момент. Другими типами контроллеров представлений являются контроллеры табличных представлений (table view controllers), отображающие ваши данные в табличной форме, выборщики изображений (image pickers), позволяющие пользователю выбирать изображения из библиотеки, и контроллеры выборщиков (picker controllers), запрашивающие ввод данных с помощью набора колесиков прокрутки.

Корректно написанное приложение состоит из одного окна, по крайней мере, одного контроллера представлений и хотя бы одного представления, инкапсулированного внутри этого контроллера.

Существующие шаблоны

По умолчанию каждое простое приложение, создаваемое с помощью Xcode, использует шаблон, созданный Interface Builder. Этот шаблон служит основным источником информации о размещении главного окна приложения и последующей иерархии окон. Он определяет ключевое окно, которое видят пользователи при начале работы вашего приложения, а также позволяет вам определять, какие действия инициируют переходы к различным представлениям. Xcode может построить иерархию для вашего пользовательского интерфейса без единой строчки кода!

Чтобы начать создание вашего пользовательского интерфейса, воспользуйтесь Xcode для создания нового оконного приложения (window-based) под названием Example. Будет создан каталог Example, содержащий скелет вашего приложения.

Каждое простое приложение создается с использованием шаблона MainWindow. Этот шаблон представляет объект главного окна, который загружается и отображается при начале работы приложения. Имя шаблона назначается вашему приложению из файла Info.plist, находящемуся в разделе Resources вашего проекта Xcode. Шаблон MainWindow задается как имя файла, но без расширения xib:

```
<key>NSMainNibFile</key>
<string>MainWindow</string>
```

Щелкните по папке Resources внутри вашего проекта. Вы увидите файл MainWindow.xib. Это файл шаблона Interface Builder. Чтобы открыть его в Interface Builder, дважды щелкните по этому файлу.

Новые шаблоны

Чтобы создать новый проект Interface Builder из шаблона, выберите пункт **New** из меню **File**. Появится диалоговое окно, предлагающее вам выбрать тип шаблона, который вы хотите создать (рис. 2.1). Шаблоны Interface Builder будут автоматически созданы для вас и помещены в каталог вашего проекта.

Класс шаблонов **Cocoa Touch** специально предназначен для приложений iPhone. После того как вы его выберете, вам станут доступны следующие типы шаблонов.

- **Application**. Создает шаблон приложения, включая делегата основного приложения и главное окно. Делегат приложения будет получать уведомления при запуске приложения, его приостановке или завершении. Главное окно для отображения представления на экране предоставляет порт основного представления. Если ваше приложение будет иметь только один экран, то используйте именно этот шаблон. Чтобы поместить представление поверх окна, вам потребуется создать представление, о чем будет рассказано далее.
- **View**. Создает единственное представление внутри приложения. Используйте этот шаблон, если вы создаете для окна приложения единственный

экран представления или для каждого представления внутри приложения, имеющего несколько представлений.

- **Window**. Шаблон исключительно для создания окон. Используйте его, если вы будете вручную размещать ваши представления на единственном экране. Окна не могут хранить отдельные элементы управления, поэтому если вы планируете использовать их, то сначала вам потребуется создать представление.
- **Empty**. Пустой шаблон, позволяющий создавать с нуля все, что пожелаете.

Рис. 2.1. Окно выбора шаблона Interface Builder

Элементы пользовательского интерфейса

При открытии шаблона Interface Builder появляется окно, похожее на экран iPhone, — небольшое окно документа, представляющего логические свойства

класса и отношения к различным объектам, а также библиотеку разных элементов управления пользовательского интерфейса, имеющихся в iPhone. Воспользуйтесь окном библиотеки для перетаскивания графических элементов непосредственно на окно iPhone. Вы сможете проектировать любые требуемые представления путем перетаскивания и изменения нужных элементов. С помощью мыши вы можете менять их размеры, корректировать их размещение. Для каждого элемента пользовательского интерфейса имеется описание (рис. 2.2).

Рис. 2.2. Библиотека Interface Builder

Библиотека разбита на категории элементов, определенные типы элементов могут быть применены только к определенным объектам. Далее описаны типы категорий.

Контроллеры

Объекты контроллеров являются классами контроллеров представлений, которые управляют объектами представления данных (описываемых далее). В Interface Builder вы можете создать различные контроллеры представлений для управления навигацией, отдельными представлениями данных, панелями вкладок, таблицами или другими элементами. Контроллер представлений не может быть частью существующего представления, поэтому для создания объектов такого типа перетащите данный элемент в ваше окно документа (это небольшое окно, содержащее свойства **File's Owner** и **First Responder**). Тем самым вы создадите новое окно для контроллера, куда вы сможете перетаскивать дополнительные элементы пользовательского интерфейса.

Представления данных

Представления данных (data views) являются отдельными классами представлений, которые отображают информацию пользователю. Они включают в себя текстовые представления (text views), просмотр изображений (image viewers), Web-представления (web views) и множество других типов объектов. Представление данных управляется его контроллером представлений, который может визуализировать различные представления данных и связывать отображаемое им представление с вашими данными. К существующему представлению вы можете добавить столько представлений данных, сколько вам захочется, пользуясь руководствами по их желаемому размещению и определению необходимых размеров. Представления данных также могу содержать элементы управления и элементы навигации.

Любые объекты, добавляемые в представление данных, будут отображаться только в этом представлении. Например, панель навигации, добавленная к контроллеру представлений, останется видимой до тех пор, пока пользователь перемещается по представлениям, принадлежащим данному контроллеру представлений, а панель навигации, добавленная к представлению данных, будет видима только при отображении данного конкретного представления.

Ввод данных и значения

Когда пользователю необходимо обеспечить возможность ввода каких-либо данных, используется объект ввода (input object). Чаще всего эти объекты называются элементами управления (controls) и включают в себя переключатели, текстовые поля, сегментированные элементы управления и другие объекты, о которых вы узнаете в *главе 10*. Любой элемент управления вы можете добавить к любому представлению, и данное представление будет, вообще говоря, обрабатывать события, генерируемые при использовании данного элемента управления.

Окна, представления и панели

Помимо контроллеров и представлений данных к существующему представлению вы можете добавить дополнительные объекты окна, представления и панели. Например, чтобы добавить панель поиска, которая будет видна исключительно внутри данного представления, перетащите ее на ваш стандартный класс представления, а не на управляющее представление. О том, как работать с панелями поиска и другими подобными объектами, вы узнаете в *главе 10*.

Inspector

Interface Builder включает в себя инструмент инспектирования — Inspector, который используется для настройки поведения создаваемых вами объектов. Inspector позволяет вам менять атрибуты объекта, имеющиеся у него связи с другими ресурсами и файлами, размер объектов и, что наиболее важно, действия, которые он предпринимает в ответ на события, например, на ввод данных пользователем. Чтобы отобразить Inspector, в меню **Tools** выберите команду **Inspector**.

Разработка пользовательского интерфейса

Существуют две причины для разработки вашего первого пользовательского интерфейса с нуля. Во-первых, вы сможете получить представление о том, как работает Interface Builder. А во-вторых, вы сможете четко понять, как

различные компоненты пользовательского интерфейса соотносятся друг с другом. Существуют окна, контроллеры представлений, представления данных, элементы управления вводом и целая совокупность взаимоотношений, которые связывают их друг с другом. В данном разделе вы с нуля построите простейший интерфейс и установите его в несложное приложение.

Запустите Interface Builder и отредактируйте шаблон MainWindow, созданный с помощью вашего оконного приложения. Ваше окно документа будет содержать практически пустой шаблон с единственным объектом окна.

Окно

Объект *окна* (window) — это основная поверхность, к которой вы прикрепляете ваш контроллер представлений. Окно представляет собой пространство на физическом экране iPhone.

В окне документа выберите значок **Window** и откройте Inspector, нажав **Command-1** или выбрав команду **Inspector** из меню **Tools**. Inspector предоставит вам параметры для изменения цвета и других свойств окна. Если ваше приложение будет использовать множественные жесты, то их применение также может быть разрешено отсюда.

Контроллер представлений

Перетащите Tab Bar Controller в окно документа. Tab Bar Controller — это тип контроллера представлений, который предоставляет набор вкладок вдоль нижнего края окна. Пользователь может щелкать по различным вкладкам, отображая разные представления, поддерживаемые вашим приложением. После перетаскивания контроллера в окно документа вы увидите, что к вашему шаблону добавился соответствующий значок. Дважды щелкните по этому значку, появится новое окно, содержащее две вкладки и серое окно с текстом "View" посередине. Кнопки представляют вашу панель вкладок, а серое окно — первую страницу представления вашего приложения.

Если вы поместите указатель мыши над названием вкладки, то сможете щелкнуть по нему, чтобы переименовать вкладку. Либо вы можете выбрать вкладку из списка предустановленных вкладок, щелкнув по вкладке кнопок (button tab) и открыв Inspector. В Inspector вы можете задать идентификатор вкладки кнопок, например, Recents, History и т. д. Вы можете определить и собственный идентификатор. Идентификатор указывает на вкладку, чтобы

вы могли ссылаться на нее из вашего кода. На кнопку вы сможете также добавить свое изображение и установить его размер. Чтобы настроить обе вкладки кнопок, воспользуйтесь Inspector.

> ***Примечание***
>
> Многие из свойств, которые вы можете установить с помощью Interface Builder, можно воспроизвести и в коде. Рассматривайте Interface Builder как GUI для кода. Хотя Interface Builder и поддерживает многие из характеристик, которые вы можете определить в коде, он не поддерживает их все. Кодирование таких объектов вы начнете изучать в *главе 3*.

На данный момент вы покончили с представлением контроллера и объектами окна. Сохраните шаблон MainWindow. Вы вернетесь к нему позже для внесения финальных корректировок.

Представления

К настоящему моменту вы настроили в вашем управляющем представлении вкладки кнопок и теперь готовы создать два отдельных представления, чтобы поставить их в соответствие каждой кнопке. Чтобы ваш интерфейс оставался модульным, мы предлагаем вам создать отдельный класс представления для каждой вкладки и использовать разные файлы Interface Builder для описания каждой из них. Вы сохраните шаблон Interface Builder для представления в отдельном файле, и он будет автоматически загружаться, когда пользователь будет переключать вкладки.

Создайте новый шаблон **Cocoa Touch View**. В отличие от шаблона приложения этот шаблон вместо объекта окна содержит объект класса представления. Это представление определяет, что отображается пользователю, когда он щелкает по первой вкладке.

В окне библиотеки прокрутите до раздела **Data Views** и перетащите объект **Table View** на ваше представление. Убедитесь в том, что он отцентрован по верху окна и его размер позволяет использовать весь экран. Щелкните по таблице и откройте Inspector. В нем вы можете поменять множество различных свойств, относящихся к внешнему виду и поведению данной таблицы. Сделав все необходимые изменения, сохраните этот файл под именем Recents.

Создайте второй проект **Cocoa Touch** — типа **View** и на этот раз перетащите в окно текстовое представление. После настройки в Inspector свойств текстового представления сохраните этот файл под именем History.

Соединение представлений

На данный момент вы создали окно, управляющее представление и два представления данных для отображения в них данных. Все, что осталось, — это соединить управляющее представление с отдельными представлениями данных так, чтобы при касании пользователем одной из кнопок отображалось соответствующее представление.

Рис. 2.3. Настроенная панель вкладок контроллера представлений

Откройте ваш шаблон MainWindow. Щелкните по первой вкладке, чтобы сделать ее активной. Теперь щелкните по серой части окна с меткой **View**. Откройте Inspector и в поле **NIB Name** введите имя файла Recents без какого-

либо расширения. В сером окне представления теперь будет написано "Loaded from Recents.nib". Теперь щелкните по второй вкладке и сделайте то же самое, но использовав имя файла History. Ваш итоговый шаблон должен быть похожим на пример, показанный на рис. 2.3.

Добавление связывания в код

После завершения редактирования вам необходимо будет добавить два только что созданных файла, Recents.xib и History.xib, в проект Xcode. Перетащите их в папку Resources в вашем проекте и следуйте подсказкам, чтобы добавить их в качестве ресурсов.

Если вы взглянете на прототип класса-делегата приложения проекта, то найдете целый ряд свойств, использующих директиву `IBOutlet`. Эта директива уведомляет Interface Builder о наличии аутлетов в вашем коде, чтобы он мог связать объекты Interface Builder с именами переменных, заданных в вашем коде — в данном случае `window` и `viewController`:

```
@property (nonatomic, retain) IBOutlet UIWindow *window;

@property (nonatomic, retain) IBOutlet RootViewController
*viewController;
```

Внутри вашего класса-делегата окно, контроллер представлений и все другие объекты, созданные вами с помощью Interface Builder, синтезируются из шаблонов, заданных вами в файле NIB:

```
@synthesize window;

@synthesize viewController;
```

Директива `synthesize` создает необходимое связывание из Interface Builder в свойства, представляющие объекты в вашем коде, а доступ к ней может осуществляться как к переменным в вашем классе. Изнутри они обрабатываются, как если бы они были переменными, созданными вами самими:

```
[ window addSubview: viewController.view ];

[ window makeKeyAndVisible ];
```

Снаружи любой другой класс, содержащий указатель на данный класс, может ссылаться на них:

```
MyAppDelegate *appDelegate;

[ appDelegate.window addSubview: appDelegate.viewController.view ];
```

Объект из главного шаблона будет загружаться подобным образом при запуске приложения, но вы так же можете породить некоторые объекты напрямую из вашего кода. К примеру, чтобы создать новый объект контроллера представлений, включите шаблон MyController.xib путем добавления его в папку Resources вашего проекта. Породите класс, используя метод `initWithNibName` класса `UIViewController`:

```
MyApplicationViewController *myViewController = [
    [ MyViewController alloc ]
    initWithNibName: @"MyController"
    bundle: nil
];
```

Вы сможете получить доступ к данной переменной тем же способом, которым вы синтезировали объекты:

```
[ window addSubview: myViewController.view ];
```

Более подробно о создании классов контроллеров представлений и работе с ними вы узнаете в *главе 3*.

Удаление Interface Builder из проекта

Если вы не собираетесь использовать Interface Builder в каком-либо конкретном проекте, вы можете полностью удалить поддержку его из проекта. Чтобы выполнить это, последуйте инструкциям, приведенным далее. После выполнения этих шагов ваш проект больше не будет использовать nib-файлы Interface Builder для получения любой содержащейся в них информации о заданном по умолчанию расположении элементов вашего пользовательского интерфейса. Это означает, что вам придется самостоятельно программировать поддержку окон, представлений и других объектов. В следующей главе приведено руководство по этим объектам. Тем не менее, у вас останется возможность порождать объекты из nib-файлов с помощью специальных nib-методов инициализации, о которых вы узнаете позже.

Примечание

Использование Interface Builder является вопросом личных предпочтений. Многим разработчикам он нравится, поскольку позволяет создавать пользовательские интерфейсы в среде WYSIWYG. Другие же считают его неудобным и предпочитают описывать все свои объекты пользовательского

интерфейса в своем исходном коде. Некоторая функциональность Interface Builder не лишена ошибок, поэтому вам захочется знать, как создавать те или иные объекты, если вы столкнетесь с подобными проблемами.

1. Откройте ваше новое оконное приложение и раскройте в проекте папку Resources. Вы увидите один или более файлов с расширением xib, например, MainWindow.xib. Щелкните правой кнопкой мыши по каждому такому файлу и в появившемся контекстном меню выберите команду **Delete**. Тем самым вы удалите шаблоны Interface Builder из проекта.
2. В папке Resources вы также увидите файл Info.plist. Щелкните по этому файлу, чтобы открыть его. Откроется окно свойств, содержащее информацию о вашем приложении. Внизу вы увидите ключ свойства **Main nib file base name**. Щелкните по этому свойству и нажмите клавишу <Delete>, чтобы удалить его из вашего проекта, затем сохраните ваши изменения. Это приведет к тому, что ваше приложение будет загружаться, не требуя файла Interface Builder.
3. Теперь, когда вы отсоединили Interface Builder от вашего приложения, вам придется вручную указать имя класса-делегата приложения. Это класс, чей метод `applicationDidFinishLaunching` будет уведомляться при запуске вашего приложения, и является тем местом, где начинается код вашего пользовательского интерфейса. Имя класса-делегата вашего проекта вы можете найти внутри папки Classes в вашем проекте. Его имя — это имя вашего проекта с суффиксом AppDelegate. Например, если вы назвали ваш проект Wizbang, то именем вашего класса-делегата будет WizbangAppDelegate. Раскройте папку Other Sources в вашем проекте и отредактируйте файл main.m. Отредактируйте строку `UIApplicationMain`, чтобы указать имя вашего класса-делегата. Например:

   ```
   int retVal = UIApplicationMain(argc, argv, nil,
   @"WizbangAppDelegate");
   ```

   Кроме того, из вашего класса можете удалить директивы `property` и `synthesize`. В нашем примере мы оставили их на случай, если вы захотите, чтобы они работали с шаблонами Interface Builder.

Глава 3

Введение в UI Kit

UI Kit — самая большая библиотека iPhone с точки зрения размера файла, 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 библиотеке UI Kit, что придает интерфейсу iPhone изысканность и единообразие. Для всех приложений iPhone доступны одни и те же программные интерфейсы UI Kit API, поэтому понимание того, как пользоваться этой библиотекой, позволит вам применять все те инструменты, с помощью которых собственные приложения Apple становятся такими эффектными.

UI Kit — это больше, чем просто набор инструментов пользовательского интерфейса; это также фундамент для GUI-приложений iPhone. При запуске приложения его функция `main` обрабатывает объект `UIApplication` внутри UI Kit. Этот класс — базовый для всех приложений, имеющих в iPhone пользовательский интерфейс, и он предоставляет доступ уровня приложений к функциям iPhone более высокого уровня. Помимо этого, общие сервисы уровня приложения, например, передача элемента управления другим приложениям и возврат к активному состоянию, являются функциями класса `UIApplication`.

Все приложения iPhone, создаваемые в Xcode, по умолчанию подключены к библиотеке UI Kit, поэтому вам не нужно ничего делать специально, чтобы подключить ее к вашему приложению. Чтобы создать скелет для примеров приложений UI Kit, приводимых в этой главе, вам необходимо будет исполь-

зовать Xcode для создания нового *оконного приложения* (window-based), *приложения на базе представления* (view-based) или *навигационного приложения* (navigation-based). Каждое из них использует различный набор файлов, для которых мы предоставим соответствующий код.

Хотя существует несколько компонентов UI Kit, которые можно создать с помощью Interface Builder, многие разработчики предпочитают создавать объекты пользовательского интерфейса непосредственно внутри своего кода. Более того, ваше приложение может стать гибридом между стандартными объектами, созданными Interface Builder, и собственными классами представлений. Эта глава описывает основные объекты UI Kit (т. е. наиболее фундаментальные для большинства приложений) и поможет вам выбрать, какие из них программировать самостоятельно, а какие создавать с помощью Interface Builder. Даже если вы решите использовать Interface Builder для создания всего вашего пользовательского интерфейса, эта глава покажет вам, как работать с созданными вами объектами.

Основные элементы пользовательского интерфейса

Целью этой главы является подготовка вас к работе с основными составляющими пользовательского интерфейса UI Kit. Более сложные функциональные возможности UI Kit описаны в *главе 10*. Вот основные компоненты.

- *Окна, представления и контроллеры представлений.* Окна и представления являются основными классами для создания пользовательского интерфейса любого типа. Окно (window) представляет собой геометрическое пространство на экране, в то время как класс представления (view) насыщает это пустое пространство своей функциональностью. Все небольшие компоненты пользовательского интерфейса, например, панели навигации, кнопки и текстовые поля, привязаны к классам представления, а уже само представление прикрепляется к какому-либо окну.

  Контроллер представлений (view controller) — класс контроллера, который управляет представлением и тем, как оно визуализируется на экране. Это придает представлению дополнительную функциональность, например, встроенную поддержку поворотов экрана, переходов и других событий экрана.

- *Текстовые представления* (text views) — это специализированные классы представлений, предоставляющие окна редактирования для просмотра или редактирования текста. Очень хорошим примером простейшего текстового представления является приложение Блокнот (Notepad). Вообще говоря, текстовые представления считаются весьма простыми и редко используются в более эффектных классах UI Kit, но являются вполне подходящим началом для знакомства с UI Kit.

- *Панели навигации и контроллеры.* Пользовательский интерфейс iPhone рассматривает различные экраны, как если бы они были страницами в книге. Панели навигации (navigation bars) зачастую служат для предоставления пользователю визуального напоминания, которое позволяет пользователю вернуться назад к предыдущему представлению, обеспечить его кнопками для изменения элементов на текущей странице экрана и отобразить набор элементов управления, например, сегментированные элементы управления и панели инструментов. Панели навигации можно обнаружить практически в каждом изначально загруженном приложении iPhone.

  Контроллеры навигации (navigation controllers) могут управлять навигацией для различных контроллеров представлений в том смысле, что контроллеры представлений могут проталкиваться в стек представлений и выталкиваться из него, оставляя всю работу по изменению панели навигации контроллеру. Каждый контроллер представлений владеет собственным набором свойств панели навигации, которые отображаются контроллером навигации в тот момент, когда данное представление становится активным.

- *Переходы.* Согласно принципам дружественности пользовательских интерфейсов, Apple в iPhone были введены переходы окон (window transitions), чтобы создать у пользователей ощущение перемещения по их приложению словно по страницам книги. Для таких визуальных переходов от одного представления к другому вместо простого мигания экрана используется анимация.

- *Представления предупреждений и листы действий.* В iPhone аналогом всплывающих окон с предупреждениями являются представления предупреждений (alert views) и листы действий (action sheets). Эти объекты появляются как модальные окна, выскальзывающие снизу на передний план экрана в тех случаях, когда какая-либо операция требует привлечения внимания пользователя. Чаще всего их можно увидеть на предустановленных на iPhone приложениях при получении пользователем какого-

либо предупреждения (например, текстового сообщения) или при попытке пользователя удалить какие-либо элементы (скажем, голосовую почту). Листы действий могут быть запрограммированы задавать пользователю любые вопросы и предоставлять ему для ответа любое количество кнопок. Они весьма полезны в тех частях приложения, которые требуют немедленной реакции.

- *Табличные представления и контроллеры.* Табличные представления (table views) на самом деле являются списками, которые можно использовать для отображения файлов, сообщений или других типов коллекций. Они служат для выделения одного или нескольких элементов способом, используемым в списках. Объекты таблицы очень гибки и позволяют разработчику определять внешний вид и поведение ячейки таблицы. Вы можете использовать таблицы для отображения простых списков, сгруппированных настроек или списков разделов в стиле старой картотеки (Rolodex). В этой главе описаны простые таблицы, а о более расширенном использовании таблиц вы узнаете в *главе 10*.

  Контроллер табличного представления (table view controller) управляет объектом табличного представления и может включать в себя добавленную поддержку контроллера представлений для табличного представления. Контроллеры табличного представления предоставляют автоматическую обработку поворотов экрана и других событий, а для упрощения навигации вы можете протолкнуть контроллер представлений в стек навигации точно так же, как и другие контроллеры. Кроме того, контроллеры табличных представлений могут выступать в роли источников данных для передачи таблице информации для отображения.

- *Действия со строкой состояния.* Строка состояния (status bar) — это небольшая панель, расположенная наверху экрана iPhone и отображающая время, уровень зарядки батареи и качество сигнала. Стиль, прозрачность и другие свойства строки состояния можно менять и настраивать.

- *Бейджи приложений.* Те приложения, которым необходимо в определенные промежутки времени уведомлять пользователя о каких-либо событиях, имеют возможность отображать на домашнем экране iPhone (springboard) бейджи (badges). Это позволяет проинформировать пользователя о том, что некоторое 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 или 3G.

- *Сервисы приложений.* Когда приложение перестает быть активным, возобновляет или прекращает свою работу, для незамедлительной обработки очистки или сохранения состояния уведомляются различные методы приложения. Уведомления отправляются делегату приложения, который затем в ответ может произвести очистку или сохранение важной информации.

Окна и представления

Основным компонентом пользовательского интерфейса является объект `UIWindow`. Окно предоставляет подложку для отображения информации внутри вашего приложения. Оно выступает в роли рамки картины, к которой вы можете прикрепить содержимое. Если окна, которые вы создаете в настольных системах, имеют строки заголовков, рамки и кнопки, объект iPhone `UIWindow` не имеет никаких визуальных компонентов: это исключительно смотровое отверстие. Как правило, для использования в жизненном цикле вашего приложения вы будете создавать только один объект `UIWindow` и прикреплять к нему один или более объектов, порожденных от `UIView`.

`UIView` является базовым классом, созданным для обеспечения прорисовки объектов в окнах. Если рассматривать `UIWindow` как рамку картины, то класс `UIView` можно рассматривать как полотно этой картины. Для создания визуальных объектов, отображающих текстовые поля, таблицы и Web-страницы, от `UIView` порождены многие классы. Этот основной класс представляет общий уровень функциональности, предоставляющий основные жесты, процедуры прорисовки и реагирования. Вы сможете создавать собственные классы на базе `UIView` для визуализации объектов своих типов или прикреплять к объекту `UIView` другие компоненты пользовательского интерфейса для их отображения.

Если в своем проекте вы используете Interface Builder, то при старте приложения окно и представление могут быть автоматически сгенерированы для вас. Инструкции для этого находятся в nib-файле Interface Builder, который выступает в роли потока ресурсов (resource fork) для вашего пользовательского интерфейса. Если вы не используете Interface Builder, то вам придется создавать эти объекты непосредственно в вашем коде.

Создание окна и представления

Прежде чем вы сможете что-либо отобразить на экране iPhone, вы должны создать окно для хранения содержимого, а чтобы построить окно, вам необ-

ходим *фрейм* (frame). Фрейм — это прямоугольная область на экране, в которой должно отображаться ваше содержимое. Структурой, содержащей эту информацию, является `CGRect`. Структура `CGRect` содержит два элемента: координаты левого верхнего угла окна (`origin`) и ширину и высоту окна (`size`). Каждый объект, который может отобразить себя на экране, имеет фрейм, задающий его область отображения, хотя многие классы высокого уровня автоматически рассчитывают его за вас. Другие же автоматически задаются при инициализации объекта представления посредством метода инициализации `initWithFrame`.

При создании основного окна координаты фрейма отсчитываются от самого экрана. Однако координаты всех последующих объектов отсчитываются от объекта, к которому они прикреплены. Например, координаты фрейма, прикрепленного к окну, отсчитываются относительно данного окна, а не экрана. Координаты объектов, прикрепленных к представлению, отсчитываются относительно данного представления и т. д.

Приложение использует при отображении весь экран iPhone, поэтому окну необходимо поставить в соответствие набор координат, отражающих область всего экрана. Для этого внутри класса `UIScreen` существуют два метода.

Метод `boundes` возвращает границы всего экрана, включая пространство, используемое строкой состояния:

```
CGRect screenBounds = [ [ UIScreen mainScreen ] bounds ];
```

Метод `applicationFrame` возвращает ту часть экрана, на которой возможно отображение вашего приложения. Это пространство не включает строку состояния:

```
CGRect screenBounds = [ [ UIScreen mainScreen ] applicationFrame ];
```

Эта область, присвоенная структуре `CGRect` объекта `screenBounds`, затем используется для создания и инициализации нового объекта `UIWindow`:

```
self.window = [ [ UIWindow alloc ] initWithFrame: screenBounds ];
```

Итак, теперь создан фрейм окна, но он ничего не содержит и не имеет никаких указаний насчет отображения чего-либо — это всего лишь невидимый объект на экране. Необходим объект, который сможет визуализировать внутри рамки содержимое, а поэтому для заполнения окна вам нужен объект, основанный на классе `UIView`.

Как вы только что узнали, координаты окна отсчитываются относительно экрана, а координаты представления — относительно окна, к которому оно прикреплено. В то время как начало координат окна на экране начинается в

пикселе, расположенном под строкой состояния (0, 20), начало координат представления внутри этого окна располагается в точке (0, 0), означающей левый верхний угол данного окна. Чтобы покончить с этим, вам необходимо получить границы `applicationFrame` и установить вертикальный отступ так, чтобы он стал нулем для окна:

```
CGRect viewBounds = [ [ UIScreen mainScreen ] applicationFrame ];
viewBounds.origin.y = 0.0;
```

Теперь вы будете использовать `viewBounds` для задания отступа и измерения класса представления:

```
UIView *myView = [ [ UIView alloc ] initWithFrame: viewBounds ];
```

> ***Примечание***
>
> Этот код немного изменится после того, как вы узнаете больше о контроллерах представлений. При использовании контроллеров представлений окно учитывает границы всего экрана, включая строку состояния, а представление — фрейм приложения, как он есть. Это позволяет контроллеру представлений корректно обрабатывать повороты экрана.

Отображение вида

Вы создали пару — окно и представление, но ни то, ни другое еще не отображено на экране. Для этого прикрепим представление к окну как вложенное представление:

```
[ window addSubview: myView ];
```

Теперь переведите окно на передний план и отобразите его с помощью метода `makeKeyAndVisible` класса `UIWindow`:

```
[ window makeKeyAndVisible ];
```

Самое бесполезное приложение: HelloView

Прежде чем мы перейдем к HelloWorld, поэкспериментируем с еще более бесполезным приложением — HelloView. Это приложение не делает ничего, кроме как просто создает пару — окно и представление. На самом деле, поскольку класс `UIView` является всего лишь базовым классом, то он не может отобразить даже текст. Все, что вы увидите, — это черный экран. Единствен-

ное, чем это приложение полезно, — это несколько первых строк кода, которые будет использовать любое приложение GUI на iPhone.

Вы можете скомпилировать это приложение, представленное в листингах 3.1—3.3, с помощью SDK, создав проект оконного приложения HelloView. Как и в большинстве примеров этой главы, вы можете пожелать удалить Interface Builder из проекта, чтобы получить представление о том, как каждый объект в отдельности создается в коде.

Листинг 3.1. Прототипы делегата приложения HelloView (HelloViewAppDelegate.h)

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

@interface HelloViewAppDelegate : NSObject <UIApplicationDelegate> {
  /* Единственное окно в вашем приложении */
  UIWindow *window;

  /* Класс представления, который вы будете отображать в окне */
  UIView *myView;
}

@property (nonatomic, retain) IBOutlet UIWindow *window;

@end
```

Листинг 3.2. Делегат приложения HelloView (HelloViewAppDelegate.m)

```
#import "HelloViewAppDelegate.h"

@implementation HelloViewAppDelegate

@synthesize window;

- (void)applicationDidFinishLaunching:(UIApplication *)application {
    CGRect screenBounds = [ [ UIScreen mainScreen ] applicationFrame ];
    CGRect windowBounds = screenBounds;
    windowBounds.origin.y = 0.0;
```

```
    /* Инициализируем окно */
    self.window = [ [ UIWindow alloc ] initWithFrame: screenBounds ];

    /* Инициализируем представление */
    myView = [ [ UIView alloc] initWithFrame: windowBounds ];

    /* Прикрепляем представление к окну */
    [ window addSubview: myView ];

    /* Делаем окно ключевым и видимым */
    [ window makeKeyAndVisible ];
}
- (void)dealloc {
    [ myView release ];
    [ window release ];
    [ super dealloc ];
}

@end
```

Листинг 3.3. Функция `main` для HelloView (main.m)

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

int main(int argc, char *argv[]) {
  NSAutoreleasePool * pool = [[NSAutoreleasePool alloc] init];

  /* Вызываем с именем класса-делегата нашего приложения */
  int retVal = UIApplicationMain(argc, argv, nil,
                                 @"HelloViewAppDelegate");
  [pool release];
  return retVal;
}
```

Как это работает

На первый взгляд приложение HelloView не делает ничего особенного, но вот его внутренняя работа:

1. При запуске приложения вызывается его функция `main()`, точно так же, как это происходит в обычной программе на C. Происходит переход в страну Objective-C и порождение приложения. Затем приложение уведомляет своего делегата, `HelloViewAppDelegate`, который указан в вызове `UIApplicationMain`. Функция `main` также отвечает за инициализацию автоматически освобождаемого пула (auto-release pool). Автоматически освобождаемые пулы широко применяются в библиотеке Cocoa от Apple для удаления объектов, которые при своем создании проектировались как `autorelease`. Тем самым приложению указывается просто уничтожить их после того, как отпадет надобность в них, и освободить занимаемые ими ресурсы.
2. Как только объект инициализирован, базовая библиотека вызывает метод `applicationDidFinishLaunching` класса `HelloViewAppDelegate`. Именно в этом месте зарождается жизнь приложения на Objective-C.
3. Метод `ApplicationFrame` класса `UIScreen` вызывается для того, чтобы вернуть координаты и размер отображаемого экрана приложения. Затем эти данные используются для создания нового окна, в котором будет размещаться основное представление приложения. Имейте в виду, что в дальнейшем по мере того, как ваше приложение будет становиться более совершенным, вы измените этот код для того, чтобы использовать всю область экрана.
4. Затем создается основное представление с использованием области отображения, начинающейся в точке (0, 0) — левый верхний угол окна. Представление становится содержимым окна.
5. Затем делегат приложения дает окну команду перейти на передний план и отобразиться. Тем самым отображается представление, которое на данный момент не имеет содержимого.

Порождение от класса *UIView*

Пример HelloView показал нам, как мало кода необходимо для создания и отображения пары "окно/представление". Поскольку сам по себе класс `UIView` является всего лишь базовым классом, то он ничего не отображает.

Чтобы создать полезное приложение, можно либо прикрепить к `UIView` более полезные объекты, либо породить новый класс `UIView`, добавив тем самым функциональности. В iPhone SDK имеется много подклассов класса `UIView`, предлагающих различные типы функциональности. Но вы можете сформировать и собственный подкласс для создания собственного представления.

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

```
@interface MainView : UIView
{
}

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

@end
```

Класс `MainView` наследует от класса `UIView`, поэтому в данный момент он делает то же, что и класс `UIView`, т. е. ничего. Чтобы сделать этот класс полезным, вам необходимо добавить в него функциональность. Далее представлена новая версия приведенного выше фрагмента кода, в которой добавлено хранение переменной для текстового поля `UITextView`. На самом деле вы можете легко прикрепить объект `UITextView` к самому окну, поскольку он порожден от класса `UIView`. Но сейчас вы включите его в ваш собственный класс:

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

@interface MainView : UIView
{
  UITextView *textView;
}
- (id)initWithFrame:(CGRect) rect;
- (void)dealloc;

@end
```

В приведенном фрагменте кода обратите внимание на два метода: `initWithFrame` и `dealloc`. Эти методы являются заданными по умолчанию ме-

тодом-инициализатором и методом-деструктором и будут подменены в вашем подклассе `MainView` с целью расширения функциональности класса `UIView`.

> ***Примечание***
>
> Классы используют общий конструктор `alloc`, но вам не нужно подменять эти методы. Ваш собственный код инициализации подходит для методов-инициализаторов, таких как `initWithFrame`.

Как вы видели, метод `initWithFrame` вызывается, когда представление порождается впервые, и используется для инициализации класса. В него передается фрейм, чтобы описать его область отображения. Внутри данного метода вам необходимо поместить любой код, инициализирующий переменные или другие объекты. Второй метод, `dealloc`, вызывается, когда устраняется данный объект. Здесь вы должны освободить все ресурсы, занятые ранее внутри вашего класса, чтобы все они были освобождены при разрушении объекта. Оба метода вызывают методы своего суперкласса, чтобы разрешить классу `UIView` обрабатывать собственные внутренние функции.

Вот шаблоны для этих двух важных методов:

```
@implementation MainView
- (id)initWithFrame:(CGRect)rect {
  /* Сначала вызываем метод initWithFrame суперкласса
   * для инициализации объекта UIView */
  self = [ super initWithFrame: rect ];

  /* Если объект уже был инициализирован, то self
   * будет иметь значение nil */

  if (self != nil) {
    /* Здесь инициализируйте переменные вашего класса */
    /* Здесь выделите первичные ресурсы вашего класса */
  }

  return self;
}

- (void)dealloc
{
  /* Здесь освободите ресурсы вашего класса */
```

```
  /* Вызовите метод dealloc суперкласса,
   * чтобы освободить все ресурсы, удерживаемые UIView */

  [ super dealloc ];
}
@end
```

Традиционное бесполезное приложение: HelloWorld

Теперь, когда вы знаете, как породить класс `UIView`, у вас есть все, что вам нужно для написания приложения, которое хоть что-либо делает — пусть даже и нечто самое бесполезное. Согласно традиции мы отдадим дань нашим легендарным Кернигану (Kernighan) и Ричи (Ritchie) и представим формально бесполезное приложение, выводящее фразу "Hello, World!", для iPhone SDK.

Вы можете скомпилировать это приложение, приведенное в листингах 3.4—3.6, с помощью iPhone SDK, создав проект оконного приложения HelloWorld. Вы можете удалить Interface Builder из проекта, чтобы получить представление о том, как каждый объект в отдельности создается в коде.

Листинг 3.4. Прототипы делегата приложения HelloWorld (HelloWorldAppDelegate.h)

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

@interface MainView : UIView
{
  UITextView *textView;
}
@end

@interface HelloWorldAppDelegate : NSObject <UIApplicationDelegate,
  UITextViewDelegate> {
    UIWindow *window;
    MainView *myMainView;
  }

@property (nonatomic, retain) IBOutlet UIWindow *window;

@end
```

Листинг 3.5. Делегат приложения HelloWorld (HelloWorldAppDelegate.m)

```
#import <UIKit/UITextView.h>
#import <UIKit/UIColor.h>
#import <UIKit/UIFont.h>
#import "HelloWorldAppDelegate.h"

@implementation MainView

- (id)initWithFrame:(CGRect) rect {
    self = [ super initWithFrame: rect ];

    if (self != nil) {
      textView = [ [ UITextView alloc] initWithFrame: rect ];
      textView.text = @"Hello, World!";

      [ self addSubview: textView ];
    }

    return self;
}

- (void)dealloc {
    [ textView release ];
    [ super dealloc ];
}
@end

@implementation HelloWorldAppDelegate

@synthesize window;

- (void)applicationDidFinishLaunching:(UIApplication *)application {
    CGRect screenBounds = [ [ UIScreen mainScreen ] applicationFrame ];
    CGRect windowBounds = screenBounds;
    windowBounds.origin.y = 0.0;
```

```
    self.window = [ [ [ UIWindow alloc ] initWithFrame: screenBounds ]
        autorelease
    ];

    myMainView = [ [ MainView alloc ] initWithFrame: windowBounds ];
    [ window addSubview: myMainView ];
    [ window makeKeyAndVisible ];
}

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

@end
```

Литсинг 3.6. Функция `main` для HelloWorld (main.m)

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

int main(int argc, char *argv[]) {
  NSAutoreleasePool * pool = [[NSAutoreleasePool alloc] init];
  int retVal = UIApplicationMain(argc, argv, nil,
                                 @"HelloWorldAppDelegate");
  [pool release];
  return retVal;
}
```

Как это работает

Пример с фразой "Hello, World!" содержит все, что вы видели до этого момента, а также подкласс `MainView` класса `UIView`, который может отображать текст.

1. Приложение порождается точно так же, как и раньше, — путем вызова программной функции `main`, которая вызывает функцию `UIApplicationMain` и последующие уведомления классу-делегату `HelloWorldAppDelegate`.

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 представления.
4. `UITextView`, о котором вы узнаете более подробно в следующем разделе, создается и прикрепляется к объекту `MainView`. Этому текстовому представлению указывается отобразить текст "Hello, World!" Объект `UITextView` добавляется к объекту `MainView` как подчиненное представление, а объект `MainView` в свою очередь добавляется как подчиненное представление к объекту `UIWindow`.
5. Окну дается указание стать видимым, отобразив объект `MainView`, который отображает прикрепленный к нему объект `UITextView`.

Контроллеры представлений

В предыдущем примере вы создали собственный класс представления `MainView`, который взял на себя реализацию всей функциональности класса `UIView`, а также контролировал класс `UITextView`, способный отображать текст. Теперь представьте, что ваше приложение имеет множество различных экранов и должно переключаться между ними. Или, например, вашему приложению необходимо обеспечивать переворот в альбомный режим. В подобных случаях вам придется потратить значительное количество времени в попытках заставить ваш собственный класс управлять тем, как отображается `UITextView`, или как осуществить переход между различными текстовыми представлениями на экране. Хотя вы и можете заняться этим сами, iPhone SDK предоставляет контроллер представлений, специально предназначенный для выполнения всей этой работы вместо вас.

Класс `UIViewController` предоставляет низкоуровневую функциональность, требуемую для создания и отображения одного или более представлений, точно так же, как класс `MainView` управлял `UITextView` в предыдущем примере. Кроме того, контроллеры представлений от Apple добавляют встроенную поддержку изменения ориентации, уведомлений о недостатке памяти и плавных переходов — всего того, о чем вы узнаете в этой главе.

Помимо функциональности, предоставляемой классом `UIViewController`, существует множество других классов типа контроллеров, существенно облегчающих вашу жизнь как разработчика. В этой книге вы узнаете о них. Эти классы контроллера требуют, чтобы вы использовали класс `UIViewController` для хранения ваших представлений, поэтому теперь пришло время начать использовать их в вашем коде.

Создание контроллера представлений

Контроллер представлений работает почти так же, как объект `MainView`, созданный вами ранее. Вы создаете подкласс `UIViewController`, который наследует функциональность своего родительского класса. Прототип версии контроллера представлений класса `MainView` может выглядеть так:

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

@interface MainViewController : UIViewController {
    UITextView *textView;
}
- (id)init;
- (void)dealloc;
- (void)loadView;
@end
```

Как и в `MainView`, переменные, использованные в представлении, хранятся внутри класса контроллера. В данном примере объект `UITextView` хранится в контроллере для последующего отображения.

Когда контроллер представлений инициализируется, создателем объекта вызывается простой метод `init`, вместо использованного вами с `UIView` метода `initWithFrame`. Контроллер представлений создан для обработки изменений ориентации, поэтому для событий изменения ориентации и при первоначальном создании представлений вам потребуется переконфигурировать его базовые представления в соответствии с новыми границами экрана. В книжном режиме разрешение экрана составляет 320×480, а в альбомном режиме — 480×320.

Когда контроллер представлений создан, делается вызов метода `loadView`. Вы отвечаете за написание кода для этого метода, создающего базовый класс `UIView` и прикрепляющего данное представление к контроллеру.

По умолчанию класс `UIViewController` создает объект `UIView`, когда тот инициализируется. Этот объект отображается как заданное по умолчанию представление контроллера, и доступ к нему может быть получен путем адресации `self.view`. Когда вы добавляете собственные классы представлений к контроллеру, у вас есть на выбор два варианта: вы можете прикрепить ваши классы к существующему объекту `UIView` или полностью заменить объект представления.

Чтобы прикрепить собственный класс представления к существующему представлению, в вашем методе `loadView` создайте собственное представление, а затем воспользуйтесь методом `addSubview` для добавления его к отображаемому уровню представления. Это может быть особенно полезно, если вы отображаете вместе несколько представлений, например, табличное представление и представление выборщика (picker). В приведенном далее примере создаются и отображаются вертикально два текстовых представления:

```
- (void)loadView {

    CGRect bounds = [ [ UIScreen mainScreen ] applicationFrame ];

    textView1 = [ [ UITextView alloc ] initWithFrame:
        CGRectMake(0, 0, bounds.size.width, bounds.size.height / 2)
    ];

    textView2 = [ [ UITextView alloc ] initWithFrame:
        CGRectMake(0, bounds.size.height / 2,
                      bounds.size.width,
                      bounds.size.height / 2)
    ];

    textView1.text = @"Hello, World!";
    textView2.text = @"Hello again!";
    [ self.view addSubview: textView1 ];
    [ self.view addSubview: textView2 ];
}
```

Если вы в один момент времени отображаете только одно представление или написали собственный автономный класс представления, который сам по себе может отображать много объектов, то вы можете пожелать полностью за-

менить заданный по умолчанию объект `UIView`. Вы можете сделать это, присвоив ваш собственный объект представления `self.view`. Вот пример этого:

```
- (void)loadView {
    [ super loadView ];
    CGRect bounds = [ [ UIScreen mainScreen ] applicationFrame ];

    textView = [ [ UITextView alloc ] initWithFrame: bounds ];
    textView.text = @"Hello, World! ";
    self.view = textView;
}
```

> ***Примечание***
>
> Обратите внимание на то, что больше нет необходимости отсчитывать начало координат экрана от нулевой точки окна. Это потому, что теперь окно может быть создано с учетом всей границы экрана, и, значит, приложение может контролировать даже ту часть экрана, которую занимает строка состояния.

Метод `loadView`, вообще говоря, вызывается единожды, пока не завершено базовое представление контроллера. Если памяти недостаточно, а представление еще не было отображено, то представление будет сброшено на диск. Если это происходит, то внутри контроллера уведомляется метод `didReceiveMemoryWarning`, позволяющий вам выполнить необходимую очистку ваших собственных ресурсов. Если представление необходимо отобразить снова, то метод `loadView` вызывается повторно.

Загрузка из Interface Builder

Помимо создания ваших собственных объектов вы можете породить контроллеры представлений из шаблонов Interface Builder. Это позволит вам хранить свойства пользовательского интерфейса для класса представления в шаблоне Interface Builder и соединять их с вашим собственным классом контроллера представлений. Чтобы создать новый объект контроллера представлений, включите xib-файл, добавив его к папке Resources вашего проекта. Затем вы можете породить класс с помощью метода `initWithNibName` класса `UIViewController`:

```
MainViewController *myViewController = [
    [ MainViewController alloc ]
```

```
    initWithNibName: @"MainViewController"
    bundle: nil
];
```

Изменение ориентации

При изменении ориентации контроллер представлений проверяет, нужно ли изменить ваши объекты в соответствии с заданной ориентацией. По умолчанию метод, предоставляемый Xcode, запрещает любые изменения ориентации, отличные от книжной (portrait):

```
- (BOOL)shouldAutorotateToInterfaceOrientation:
    (UIInterfaceOrientation)interfaceOrientation
{
  return (interfaceOrientation == UIDeviceOrientationPortrait);
}
```

Для поддержки всех видов ориентаций необходимо вернуть значение `YES`. Для поддержки только определенной ориентации сравните входной аргумент хотя бы с одним из следующих перечисленных далее значений:

- `UIDeviceOrientationUnknown` — хранилище ошибок или аппаратных сбоев;
- `UIDeviceOrientationPortrait` — устройство ориентировано прямо по вертикали в книжном режиме;
- `UIDeviceOrientationPortraitUpsideDown` — устройство ориентировано вверх дном по вертикали в книжном режиме;
- `UIDeviceOrientationLandscapeLeft` — устройство повернуто против часовой стрелки в альбомном режиме;
- `UIDeviceOrientationLandscapeRight` — устройство повернуто по часовой стрелке в альбомном режиме;
- `UIDeviceOrientationFaceUp` — устройство лежит на ровной поверхности лицевой стороной вверх, как, например, на столе;
- `UIDeviceOrientationFaceDown` — устройство лежит на ровной поверхности лицевой стороной вниз, как, например, на столе.

После изменения ориентации из контроллера представлений вызывается метод `didRotateFromInterfaceOrientation`. Вы можете воспользоваться им

для настройки любых необходимых приложений в связи с изменением ориентации параметров.

```
- (void)didRotateFromInterfaceOrientation:
        (UIInterfaceOrientation)fromInterfaceOrientation
{
  /* Здесь добавьте код, следующий за изменением ориентации */
}
```

Удаление контроллера представлений

Наконец, при удалении контроллера представлений вызывается его метод `dealloc`. Он удаляет все объекты, являющиеся локальными для контроллера, а затем вызывает метод его суперкласса `dealloc`:

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

Ваша версия метода `dealloc` подменит собственную версию класса, вот почему метод `dealloc` его суперкласса вызывается позже. Все объекты, занятые вами внутри вашего подкласса, должны быть удалены в методы `dealloc`, который освободит их при удалении вашего контроллера представлений:

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

HelloWorld в стиле контроллера представлений: ControllerDemo

Этот пример построен на основе предыдущего путем реализации надлежащего использования контроллера представлений вместо `MainView`. Как вы увидите, контроллер представлений инкапсулирует текстовое представление почти точно так же, как это делал класс `MainView`, но сразу же расширяет функциональность вашего приложения, разрешая полное вращение (совместно с другими функциями, о которых вы узнаете позднее). Данный пример показывает корректное использование методов `init`, `loadView` и `dealloc` контроллера пред-

ставлений для создания основного представления, а также примеры некоторых процедур, необходимых в связи с изменениями ориентации.

Вы можете скомпилировать это приложение, представленное в листингах 3.7—3.11, с помощью SDK, создав проект оконного приложения ControllerDemo. Если вы хотите увидеть, как создавать все эти объекты с нуля, то удалите код Interface Builder.

Листинг 3.7. Прототипы делегата приложения ControllerDemo (ControllerDemoAppDelegate.h)

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


@class ControllerDemoViewController;


@interface ControllerDemoAppDelegate : NSObject <UIApplicationDelegate> {
  UIWindow *window;
  ControllerDemoViewController *viewController;
}


@property (nonatomic, retain) IBOutlet UIWindow *window;
@property (nonatomic, retain) IBOutlet ControllerDemoViewController
*viewController;


@end
```

Листинг 3.8. Делегат приложения ControllerDemo (ControllerDemoAppDelegate.m)

```
#import "ControllerDemoAppDelegate.h"
#import "ControllerDemoViewController.h"


@implementation ControllerDemoAppDelegate


@synthesize window;
@synthesize viewController;


- (void)applicationDidFinishLaunching:(UIApplication *)application {
    CGRect screenBounds = [ [ UIScreen mainScreen ] bounds ];
```

```
    self.window = [ [ [ UIWindow alloc ] initWithFrame: screenBounds ]
        autorelease
    ];

    viewController = [ [ ControllerDemoViewController alloc ] init ];

    [ window addSubview:viewController.view ];
    [ window makeKeyAndVisible ];
}

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

@end
```

Листинг 3.9. Прототип контроллера представлений ControllerDemo (ControllerDemoViewController.h)

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

@interface ControllerDemoViewController : UIViewController {
    NSString *helloWorld, *woahDizzy;
    UITextView *textView;
}

@end
```

Листинг 3.10. Контроллер представлений ControllerDemo (ControllerDemoViewController.m)

```
#import "ControllerDemoViewController.h"

@implementation ControllerDemoViewController

- (id)init {
  self = [ super init ];

  if (self != nil) {
```

```
    /* Иллюстрируем выделение некоторых объектов,
     * даже если нам это не нужно */
    helloWorld = [[ NSString alloc ] initWithString: @"Hello, World!"];
    woahDizzy = [[ NSString alloc ] initWithString: @"Woah, I'm Dizzy!"];
  }

  return self;
}

- (void)loadView {
  [ super loadView ];

  textView = [ [ UITextView alloc ] initWithFrame:
             [ [ UIScreen mainScreen ] applicationFrame ]
  ];

  textView.text = helloWorld;
  self.view = textView;
}

- (BOOL)shouldAutorotateToInterfaceOrientation:
  (UIInterfaceOrientation)interfaceOrientation
{
  return YES;
}

- (void)didRotateFromInterfaceOrientation:
  (UIInterfaceOrientation)fromInterfaceOrientation
{
  textView.text = woahDizzy;
}

- (void)viewDidLoad {
  [ super viewDidLoad ];
  /* Добавляем собственный код, следующий за загрузкой */
}

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

```
  /* Здесь добавляем собственный код, обрабатывающий недостаток памяти */
}

- (void)dealloc {
  /* Здесь освобождаются занимаемые нами объекты */

  [ helloWorld release ];
  [ woahDizzy release ];

  [ textView release ];
  [ super dealloc ];
}

@end
```

Листинг 3.11. Функция `main` для ControllerDemo (main.m)

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

int main(int argc, char *argv[]) {
  NSAutoreleasePool * pool = [[NSAutoreleasePool alloc] init];
  int retVal = UIApplicationMain(argc, argv, nil,
                                 @"ControllerDemoAppDelegate");
  [pool release];
  return retVal;
}
```

Как это работает

В примере ControllerDemo имеется все, что вы до сих пор видели, но класс `MainView` заменен контроллером представлений `ControllerDemoViewController`:

1. Приложение порождается путем вызова программной функции `main`, которая приводит к тому, что при запуске приложения уведомляется метод `applicationDidFinishLaunching` класса `ControllerDemoAppDelegate`.
2. Создается окно, учитывающее все границы экрана. Это необходимо для корректной обработки поворотов экрана.

3. Создается новый контроллер и вызывается его метод `init`. Когда представление отображается, уведомляется метод `loadView`, который создает объект `UITextView` и присваивает его видимой области приложения, используя метод `applicationFrame` класса `UIScreen`. Затем текстовое представление назначается в качестве активного представления контроллера представлений путем установки `self.view`.
4. Активное представление контроллера представлений добавляется в качестве подчиненного представления окна, а окну дается указание отобразиться. Тем самым отображается текстовое представление.
5. При повороте устройства запрашивается метод контроллера представлений `shouldAutorotateToInterfaceOrientation` на счет того, должен ли он поменять ориентацию. В примере возвращается `YES`, а внутренние методы контроллера представлений выполняют всю необходимую работу.
6. Когда завершится поворот экрана, вызывается метод `didRotateFromInterfaceOrientation` контроллера представлений, который изменяет содержимое текстового представления.

Для дальнейшего изучения

Теперь, когда у вас есть скелет для любого приложения представлений, прежде чем продолжить немного поиграйте с ним.

- Попытайтесь изменить начало координат и размер фрейма, используемые контроллером представлений и окном. Что произойдет с окном и его потомком? А как насчет того, чтобы изменить начало координат отображения `textView`?
- В заголовочных файлах вашей SDK проверьте наличие следующих прототипов: UIWindow.h, UIView.h и UIViewController.h. Вы найдете их в папке /Developer/Platforms/iPhoneOS.platform внутри каталога Headers библиотеки UI Kit.

Текстовые представления

Класс `UITextView` неявно построен на основе класса `UIView`, но его функциональность расширена для предоставления возможностей редактирования текста, прокручивания и изменения различных параметров стиля, таких как шрифт и цвет. Текстовые представления очень удобны и практичны для текстовых частей приложения, например, электронных книг, разделов програм-

мы, где нужно делать заметки, или информационных страниц. Они предоставляют возможность редактирования неструктурированной информации. Структурированная информация более эффектно отображается в других объектах UI Kit, о которых вы узнаете в этой главе и *главе 10*. Вы уже использовали класс `UITextView` в нескольких предыдущих примерах. В этом разделе вы узнаете о том, как настраивать его поведение.

Объект `UITextView` наследует от класса `UIScrollView`, являющего классом прокручивания общего назначения. Класс текстового представления наследует всю функциональность прокрутки класса `UIScrollView`, поэтому разработчик может сфокусироваться на отображении содержимого, а не на программировании панелей прокрутки. Класс `UIScrollView` наследует от базового класса `UIView`, который является, как было упомянуто в предыдущем разделе, базовым классом для всех классов представлений. Далее представлена иллюстрация иерархии классов:

`UITextView` добавляет функциональность и наследует функциональность от ...

> `UIScrollView` добавляет функциональность и наследует функциональность от ...
>
> > `UIView` является базовым классом для всех классов представлений, наследует функциональность от ...
> >
> > > `UIResponder` является фундаментальным классом для всех объектов, отвечающих на запросы.

Создание текстового вида

Поскольку `UITextView` порожден от `UIView`, то он создан по образу и подобию объектов основного вида, созданных в предыдущем разделе, т. е. с помощью метода `initWithFrame`. Если бы ваше текстовое представление было прикреплено к методу `loadView` контроллера представлений, то для определения области отображения представления вы бы использовали метод `applicationFrame` класса `UIScreen`:

```
- (void) loadView {
    [ super loadView ];
    CGRect bounds = [ [ UIScreen mainScreen ] applicationFrame ];
    textView = [ [ UITextView alloc ] initWithFrame: bounds ];
    self.view = textView;
}
```

Иначе если вы прикрепляете текстовое представление к объекту существующего представления, например, заданное по умолчанию представление контроллера, то можете задать собственный размер текстового представления. Приведенный далее пример создает текстовое представление размером 320×200 пикселов с вертикальным отступом в 100 пикселов вниз, а затем прикрепляет его к заданному по умолчанию представлению контроллера:

```
- (void) loadView {
    [ super loadView ];
    CGRect viewRect = CGRectMake(0.0, 100.0, 320.0, 200.0);
    UITextView *textView = [ [ UITextView alloc ]
        initWithFrame: viewRect ];
    [ self.view addSubview: textView ];
}
```

После создания представления можно задать целый ряд различных свойств.

Редактирование

По умолчанию текстовое представление разрешено редактировать пользователю. Если пользователь коснется области внутри текстового представления, то iPhone автоматически отобразит клавиатуру и соответствующим образом изменит размер текстового представления. Для текстовых представлений, используемых только для чтения, установите `NO` в качестве значения свойства `editable`:

```
textView editable = NO;
```

Поля

По умолчанию текст в окне выравнивается по левому краю. С помощью свойства `textAlignment` вы можете изменить это:

```
textView.textAligment = UITextAligmentLeft;
```

Для изменения выравнивания текста пользуйтесь следующими значениями:

- `UITextAligmentLeft` — текст выравнивается по левому краю (задан по умолчанию);
- `UITextAligmentRight` — текст выравнивается по правому краю;
- `UITextAligmentCenter` — текст выравнивается по центру.

Шрифт и размер

Шрифт (гарнитура) текста и кегль могут быть установлены путем присвоения объекта `UIFont` свойству `font` текстового представления. Чтобы создать объект `UIFont`, воспользуйтесь статическим методом `fontWithName`:

```
UIFont *myFont = [ UIFont fontWithName: @"Arial" size: 18.0 ];
textView.font = myFont;
```

Помимо этого, для простого создания системных шрифтов существуют еще три других статических метода:

```
UIFont *mySystemFont = [ UIFont systemFontOfSize: 12.0 ];
UIFont *myBoldSystemFont = [ UIFont boldSystemFontOfSize: 12.0 ];
UIFont *myItalicSystemFont = [ UIFont italicSystemFontOfSize: 12.0 ];
```

Перечисленные далее шрифты поставляются с iPhone, и на них можно ссылаться по имени:

- American Typewriter;
- Apple Gothic;
- Arial;
- Arial Rounded MT Bold;
- Courier;
- Courier New;
- Georgia;
- Helvetica;
- Helvetica Neue;
- Marker Felt;
- Times;
- Times New Roman;
- Trebuchet MS;
- Verdana;
- Zapfino.

Выбор гарнитуры задает отображаемый шрифт для всего текста внутри данного текстового представления. Текстовое представление не поддерживает форматированный текст (rich text).

Цвет текста

Вы можете задать цвет текста с помощью объекта `UIColor`. Класс `UIColor` предоставляет множество различных методов для простого смешивания любых цветов. Для создания цветов вы можете использовать статические методы, которые удаляются при утрате потребности в них. Цвета могут создаваться как уровни белого (white levels) с использованием композиции оттенков или RGB. Чтобы создать простой цвет RGB, задайте множество из четырех значений с плавающей запятой в диапазоне между 0,0 (0%) и 1,0 (100%) для красного, зеленого, синего и прозрачности (alpha):

```
UIColor *myWhiteTransparentColor =
  [ UIColor colorWithWhite: 1.0 alpha: 0.50 ];

UIColor *myColorHue = [ UIColor colorWithHue: 120.0 / 360.0
                                saturation: 0.75
                                brightness: 0.50
                                alpha: 1.0
];

UIColor *myColorRGB = [ UIColor colorWithRed: 0.75
                                green: 1.0
                                blue: 0.75
                                alpha: 1.0
];
```

Если вы собираетесь повторно использовать много различных объектов `UIColor`, то можете создать их экземпляры:

```
UIColor *myWhiteTransparentColor = [ [ UIColor alloc ]
    initWithWhite: 1.0 alpha: 0.50
];

UIColor *myColorHue = [ [ UIColor alloc ]
    initWithHue: 120.0 / 360.0
    saturation: 0.75
    brightness: 0.50
    alpha: 1.0
];
```

```
UIColor *myColorRGB = [ [ UIColor alloc ] initWithRed: 0.75
    green: 1.0
    blue: 0.75
    alpha: 1.0
];
```

Класс `UIColor` также поддерживает множество статических методов для создания системных цветов, которые калибруются iPhone максимально точно. Эти методы включают в себя следующие из UIColor.h:

```
+ (UIColor *)blackColor;     // 0.0 white
+ (UIColor *)darkGrayColor;  // 0.333 white
+ (UIColor *)lightGrayColor; // 0.667 white
+ (UIColor *)whiteColor;     // 1.0 white
+ (UIColor *)grayColor;      // 0.5 white
+ (UIColor *)redColor;       // 1.0, 0.0, 0.0 RGB
+ (UIColor *)greenColor;     // 0.0, 1.0, 0.0 RGB
+ (UIColor *)blueColor;      // 0.0, 0.0, 1.0 RGB
+ (UIColor *)cyanColor;      // 0.0, 1.0, 1.0 RGB
+ (UIColor *)yellowColor;    // 1.0, 1.0, 0.0 RGB
+ (UIColor *)magentaColor;   // 1.0, 0.0, 1.0 RGB
+ (UIColor *)orangeColor;    // 1.0, 0.5, 0.0 RGB
+ (UIColor *)purpleColor;    // 0.5, 0.0, 0.5 RGB
+ (UIColor *)brownColor;     // 0.6, 0.4, 0.2 RGB
+ (UIColor *)clearColor;     // 0.0 white, 0.0 alpha
```

После создания объекта `UIColor` присвойте его свойству цвета текстовое представление:

```
textView.textColor = myColorHue;
```

Поскольку текстовое представление напрямую не поддерживает обогащенный текст (rich text), то выбор цвета влияет на весь текст внутри данного представления.

Цвета из библиотеки Core Graphics

Библиотека Core Graphics широко распространена на настольных системах с Mac OS X для двумерных визуализаций и преобразований. В этой книге мы осветим некоторые ее части, но, вообще говоря, программирование с исполь-

зованием библиотеки Core Graphics само по себе является темой для целой книги. Если вы уже использовали библиотеку Core Graphics для программирования графики, то бóльшую часть ваших знаний вы сможете применить и на iPhone.

Аналогом объекта `UIColor` в Core Graphics является объект `CGColor`. Конвертировать `CGColorRef` (справочник цветов Core Graphics) в объект `UIColor` можно с помощью метода `colorWithCGColor`:

```
CGColorSpaceRef colorSpace =
    CGColorSpaceCreateWithName(kCGColorSpaceGenericRGB);
float opaqueRed[4] = { 1.0, 0.0, 0.0, 1.0 };
CGColorRef red = CGColorCreate(colorSpace, opaqueRed);
UIColor *myRed = [ UIColor colorWithCGColor: red ];
textView.textColor = myRed;
```

Задание содержимого

Текст для представления можно задать с помощью свойства `text`. Это свойство принимает аргумент `NSString`. Вот простой способ задать статический текст:

```
textView.text = @"Hello, world!";
```

Кроме того, для создания собственных строковых объектов вы можете воспользоваться множеством методов создания строки класса `NSString`:

```
int nBottles = 100;
NSString *myFormattedString = [ [ NSString alloc ]
    initWithFormat: @"%d bottles of beer on the wall", nBottles
];

textView.text = myFormattedString;
```

Также можно создать объекты `NSString` из символьных массивов в стиле C:

```
char myBottles[] = "100 bottles of beer on the wall";
NSString *myCString = [ NSString stringWithCString: myBottles ];
textView.text = myCString;
```

Чтобы получить доступ к какому-либо файлу в вашем домашнем каталоге, воспользуйтесь функцией `NSHomeDirectory` для получения уникального пути

к песочнице вашего приложения. Это путь к иерархии каталогов, о которой вы узнали в *главе 1*:

```
NSString *myFile = [ NSHomeDirectory()
    stringByAppendingPathComponent: @"Documents/file.txt"
];

NSString *myFileString = [ NSString stringWithContentsOfFile: myFile ];
textView.text = myFileString;
```

Все то же самое существует и в виде экземпляров методов, использующих в качестве префикса `initWith` вместо `stringWith`:

```
NSString *myFile = [ [ NSString alloc ] initWithFormat:
      @"%@/Documents/file.txt",
  NSHomeDirectory()
];

NSString *myFileString = [ [ NSString alloc ] initWithContentsOfFile:
myFile ];
textView.text = myFileString;
```

Отображение HTML

Для отображения содержимого в формате HTML внутри текстового представления существует недокументированный API. Поскольку он не документирован, то, вероятнее всего, Apple хотела, чтобы вместо него вы использовали класс `UIWebView`, о котором вы узнаете в *главе 10*. Для общего развития мы покажем вам вызов закрытого метода, что может оказаться полезным в целях отладки:

```
[ textView setContentToHTMLString:
    @"<HTML><BODY><B>Hello, World!</B></BODY></HTML>" ];
```

> ***Внимание!***
>
> Недокументированный API может поменяться в любой момент времени. Более того, если вы используете недокументированный API, то вашему приложению может быть отказано во включении в списки магазина iTunes.

Чтение исходного кода Web-страницы: SourceReader

Если вы хоть раз в жизни пытались изучить HTML, то, скорее всего, вы хотя бы пару раз пользовались возможностью **View Source** (Просмотр HTML-кода) вашего обозревателя Интернета. В этом примере для загрузки первой страницы Web-узла, расположенного по адресу **http://www.oreilly.com**, и отображения ее в текстовом окне с помощью `UITextView` используются объекты `NSURL` и `NSString` базовой библиотеки.

Рис. 3.1. Пример SourceReader

Вы также увидите применение объектов `UIColor` и `UIFont` и других свойств текстового представления. Данный пример будет построен на основе предыдущего примера контроллера представлений с добавлением функционально-

сти в метод `loadView` контроллера. Поскольку вся функциональность по выполнению HTTP-запросов встроена в класс `NSString`, то вы не увидите никакого кода по загрузке содержимого Web-страницы. Содержимое Web-страницы будет автоматически загружено после вызова метода `StringWithContentsOfURL` класса `NSString` (рис. 3.1).

Вы можете скомпилировать это приложение, представленное в листингах 3.12—3.16, с помощью SDK, создав проект приложения SourceReader на базе представления. Если вы хотите увидеть, как создавать все эти объекты с нуля, то удалите код Interface Builder.

Листинг 3.12. Прототипы делегата приложения SourceReader (SourceReaderAppDelegate.h)

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

@class SourceReaderViewController;

@interface SourceReaderAppDelegate : NSObject <UIApplicationDelegate> {
  UIWindow *window;
  SourceReaderViewController *viewController;
}

@property (nonatomic, retain) IBOutlet UIWindow *window;
@property (nonatomic, retain) IBOutlet SourceReaderViewController
   *viewController;

@end
```

Листинг 3.13. Делегат приложения SourceReader (SourceReaderAppDelegate.m)

```
#import "SourceReaderAppDelegate.h"
#import "SourceReaderViewController.h"

@implementation SourceReaderAppDelegate

@synthesize window;
@synthesize viewController;
```

```
- (void)applicationDidFinishLaunching:(UIApplication *)application {
  CGRect screenBounds = [ [ UIScreen mainScreen ] bounds ];

  self.window = [ [ [ UIWindow alloc ] initWithFrame: screenBounds ]
      autorelease
  ];

  viewController = [ [ SourceReaderViewController alloc ] init ];

  [ window addSubview: viewController.view ];
  [ window makeKeyAndVisible ];
}

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

@end
```

Листинг 3.14. Прототип контроллера представлений SourceReader (SourceReaderViewController.h)

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

@interface SourceReaderViewController : UIViewController {
  UITextView *textView;
}

@end
```

Листинг 3.15. Контроллер представлений SourceReader (SourceReaderViewController.m)

```
#import <UIKit/UIColor.h>
#import <UIKit/UIFont.h>
#import "SourceReaderViewController.h"
```

```
@implementation SourceReaderViewController

- (id)init {
    self = [ super init ];
    if (self != nil) {
      /* Дополнительный инициализирующий код */
    }
    return self;
}

- (void)loadView {
    CGRect bounds = [ [ UIScreen mainScreen ] applicationFrame ];

    [ super loadView ];

    textView = [ [ UITextView alloc ] initWithFrame: bounds ];

    UIColor *myBlue = [ UIColor colorWithRed: 0.0
                                green: 0.0 blue: 1.0 alpha: 1.0 ];
    textView.textColor = myBlue;

    UIFont *myFixed = [ UIFont fontWithName: @"Courier New" size: 10.0 ];
    textView.font = myFixed;

    textView.editable = NO;

    NSURL *url = [ NSURL URLWithString: @"http://www.oreilly.com" ];
    NSString *pageData = [ NSString stringWithContentsOfURL: url ];
    textView.text = pageData;

    self.view = textView;
}

- (BOOL)shouldAutorotateToInterfaceOrientation:
    (UIInterfaceOrientation)interfaceOrientation
{
    return (interfaceOrientation == UIDeviceOrientationPortrait);
}
```

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

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

@end
```

Листинг 3.16. Функция `main` для SourceReader (main.m)

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

int main(int argc, char *argv[]) {

    NSAutoreleasePool * pool = [[NSAutoreleasePool alloc] init];
    int retVal = UIApplicationMain(argc, argv, nil,
                                   @"SourceReaderAppDelegate");
    [pool release];
    return retVal;
}
```

Как это работает

Пример SourceReader содержит все, что вы до этого момента видели, и добавляет функциональное текстовое представление, включающее в себя содержимое Web-узла O'Reilly:

1. Приложение начинается со своей функции `main`, которая вызывает метод `applicationDidFinishLaunching` класса `SourceReaderAppDelegate`.
2. Создается окно, использующее все границы экрана. Далее создается контроллер представлений и вызывается его метод `init`. Затем вызывается метод `loadView` контроллера, который создает объект `UITextView` и с помощью метода `applicationFrame` класса `UIScreen` присваивает его видимой области приложения.

3. После создания текстового представления с использованием URL **http://www.oreilly.com** создается объект `NSURL`. Это требуется для создания объекта `NSString`, хранящего в себе содержимое соответствующей Web-страницы, которую класс `NSString` загружает автоматически. Затем содержимое данной страницы присваивается текстовому представлению, а активное представление контроллера представлений замещается путем установки `self.view`.
4. Активное представление контроллера представлений добавляется как вложенное представление окна, а окну дается указание отобразиться. Тем самым отображается текстовое представление.

Для дальнейшего изучения

Прежде чем продолжить изучение материала, протеституйте немного текстовое представление.

- Используя недокументированный API, измените этот проект так, чтобы отображался сам документ в формате HTML, а не исходный текст. Какого рода предупреждения компилятора вы получили и почему?
- В заголовочных файлах вашего SDK проверьте наличие следующих прототипов: UITextView.h, UIColor.h и UIFont.h. Вы найдете их в папке /Developer/Platforms/iPhoneOS.platform внутри каталога Headers библиотеки UI Kit.

Панели навигации и контроллеры

iPhone не поддерживает панели инструментов в традиционном понимании рабочего стола: в виде набора значков вдоль верхней части окна. Поскольку каждый экран приложения рассматривается как страница в книге, то Apple создала собственную версию панели инструментов для iPhone, чтобы она имела более явный книжный вид. В отличие от панелей инструментов панели навигации (navigation bars) включают в себя заголовок страницы, кнопки направлений и текстовые кнопки для контекстно-зависимых функций, таких как включение громкой связи или очистки списка элементов. Панели навигации также поддерживают элементы управления для добавления кнопок с вкладками (называемых сегментированными элементами управления — segmented controls), например, кнопки вызова **All** или **Missing** при просмотре последних вызовов. Если вам не хватает панелей инструментов, то вы можете

создать панели инструментов с собственными кнопками. Кроме того, существует небольшая библиотека стандартных системных кнопок.

Контроллеры навигации "оборачивают" один или более контроллеров представлений, тем самым позволяя вам добавлять панели навигации к существующему приложению на базе контроллера представлений, не заботясь об обработке изменений размеров окна или ориентации устройства. Элементы управления навигацией позволяют вам помещать в стек отдельные контроллеры представлений, автоматически управляя навигацией между ними. Тем самым обеспечивается плавная навигация и управление обновлением панели навигации в соответствии со свойствами навигации каждого представления.

Свойства панели навигации — типы отображаемых кнопок и элементов управления — являются свойствами каждого контроллера представлений, а не контроллера навигации. Это позволяет каждому представлению описывать собственную конфигурацию панели навигации, которую автоматически загружает контроллер представлений, когда пользователь переходит к данному представлению. Если контроллер представлений скажет: "У меня есть кнопка громкой связи", то контроллер навигации при отображении данного представления нарисует кнопку громкой связи.

Создание контроллера навигации

Вы можете создать контроллер навигации сразу же после создания хотя бы одного контроллера представлений и инициализировать его с указателем на самое верхнее представление, т. е. на корневое представление для данного приложения. Контроллер навигации ссылается на него как на *контроллер корневого представления* (root view controller). Контроллер корневого представления — это контроллер, представляющий самый низ вашего навигационного пути: основное представление вашего приложения, которое не имеет кнопки **Back** и от которого берут свое начало все остальные представления.

Чтобы создать контроллер навигации, сначала создайте класс представления, который будет служить контроллером корневого представления. Затем породите контроллер навигации с помощью его метода `initWithRootViewController`:

```
viewController = [ [ MyViewController alloc ] init ];


navigationController = [ [ UINavigationController alloc ]
    initWithRootViewController: viewController ];
```

При использовании контроллера навигации прикрепите представление контроллера навигации к окну. Контроллер представлений прикрепляется к контроллеру навигации при создании контроллера навигации, и поэтому он тоже будет отображаться при добавлении контроллера навигации к окну:

```
[ window addSubview: [ nagivationController view ] ];
```

Контроллер навигации будет автоматически визуализировать и его, и активный контроллер представлений, который станет умолчанием для контроллера корневого представления. Когда новое представление проталкивается в стек контроллера навигации, отображается новое представление до тех пор, пока пользователь не нажмет кнопку **Back** или не перейдет куда-либо еще. Чтобы втолкнуть другое представление в контроллер навигации, воспользуйтесь методом `pushViewController`. Вот пример этого:

```
[ navigationController pushViewController: myOtherViewController
        animated: YES
];
```

Кнопка **Back** к панели навигации добавляется автоматически при проталкивании представления. Название кнопки **Back** будет соответствовать названию предыдущего в стеке контроллера представлений. Когда пользователь нажимает кнопку **Back**, контроллер представлений покидает стек, а предыдущее представление снизу переносится обратно на экран.

Свойства контроллера навигации

Класс контроллера представлений, показанный ранее в этой главе, может содержать собственные свойства панели навигации. Это позволяет вам определять различные размещения панелей навигации для каждого создаваемого вами контроллера представлений. До этого момента вы имели дело с единственным контроллером представлений в приложении, но в функциональном приложении у вас будет по одному контроллеру представлений для каждого навигационного окна, каждый со своими свойствами, определяющими внешний вид панели навигации.

Свойства панели навигации, как правило, задаются при вызове метода `init` контроллера представлений:

```
- (void)init {
  self = [ super init ];
```

```
  if (self != nil) {
    /* Свойства панели навигации */

    self.title = @"My Title";
    ...
  }
  return self;
}
```

После отображения контроллера навигации вы можете изменить множество свойств в контроллере представлений — этот контроллер изменит панель навигации в соответствии с последними настройками контроллера представлений. Все изменения в панель навигации вносятся через свойства контроллера представлений.

Все видимое в панели навигации является частью объекта `UINavigationItem`. Класс `UINavigationItem` является базовым классом для всего, что прикрепляется к панели навигации, включая кнопки и другие объекты. Apple сделала этот класс закрытым (private), поэтому вы не можете получить к нему доступ напрямую. Когда каждый элемент, например, заголовок, создаваемый в следующем разделе, добавляется к панели навигации, его объект `UINavigationItem` проталкивается в нее как объект в стеке.

Задание заголовка

Навигационный заголовок отображается в виде крупного белого текста посередине панели навигации. Заголовок часто используется, чтобы проинформировать конечного пользователя, какого рода информация отображена в окне, например, "Saved Messages". Кроме того, заголовок определяет название кнопки возврата, если контроллер представлений не является корнем стека:

```
self.title = @"Inbox"
```

Кнопки, стили и действия

Кнопки можно добавлять к левой или правой стороне панели навигации. Поскольку пространство панели ограничено, то добавлять нужно кнопки только для тех функций, которые специфичны для навигации или для отображаемой страницы.

Контроллер навигации автоматически создает новый объект `UINavigationItem` каждый раз, когда осуществляется доступ к его свойству `NavigationItem`. Это позволяет вам присваивать новые объекты панели навигации, не беспокоясь о размещении и даже о том, что собой представляет класс `UINavigationItem`.

Чтобы создать новую навигационную кнопку, создайте экземпляр класса `UIBarButtonItem`. Этот класс позволяет вам задавать название кнопки, стиль и действие, вызываемое нажатием этой кнопки пользователем:

```
UIBarButtonItem *myButton = [ [ [ UIBarButtonItem alloc ]
    initWithTitle: @"Do it!"
    style: UIBarButtonItemStylePlain
    target: self
    action: @selector(doit)
] autorelease ];
```

Для инициализации кнопки используются следующие свойства:

- `initWithTitle` — название кнопки. Этот аргумент принимает `NSString`, о котором вы уже слышали. Если кнопка является текстовой, то она будет отображаться с этим текстом внутри кнопки;
- `style` — стиль кнопки. Существуют следующие стили:
  - `UIBarButtonItemStylePlain` — стиль кнопки по умолчанию; при нажатии светится;
  - `UIBarButtonItemStyleBordered` — аналогичен `UIBarButtonItemStylePlain`, но отображает окаймленную кнопку;
  - `UIBarButtonItemStyleDone` отображает синюю кнопку, означающую то, что пользователь должен коснуться ее после завершения редактирования;
- `target` — делегат для действия. Это объект, который будет получать уведомления, когда пользователь нажимает данную кнопку. Использование `self` приведет к уведомлению объекта контроллера представлений в предположении, что вы создали кнопку в контроллере представлений;
- `action` — метод, вызываемый, когда пользователь нажимает данную кнопку. Тем самым уведомляется метод, заданный в вашем целевом (target) объекте, чтобы он мог предпринять соответствующее действие.

После создания кнопки вы можете добавить ее как левую или правую кнопку панели навигации, после этого она будет незамедлительно отображена:

```
self.navigationItem.leftBarButtonItem = myButton;
self.navigationItem.rightBarButtonItem = myButton;
```

Чтобы создать кнопку со стрелкой влево (кнопка **Back**), присвойте кнопку свойству `backBarButtonItem`. Кнопка **Back** отображается только в том случае, если контроллер представлений не является контроллером корневого представления:

```
self.navigationItem.backBarButtonItem = myButton;
```

Когда кнопка нажата, на целевом объекте получает уведомление селектор. Вам необходимо написать метод, который будет обрабатывать это нажатие кнопки. В примере далее приведен селектор `doit`. Добавьте к вашему классу-делегату метод с таким же именем:

```
- (void)doit {
  /* Сюда поместите свой код */
}
```

Если в какой-либо момент вы захотите запретить одну из существующих кнопок панели навигации, то у вас есть два варианта действий. Чтобы заставить кнопку исчезнуть, воспользуйтесь свойствами кнопки, о которых вы узнали ранее, и установите кнопку в `nil`:

```
self.navigationItem.leftBarButtonItem = nil;
```

Чтобы оставить кнопку видимой, но недоступной (серой), запретите кнопку, установив `NO` в качестве значения свойства `enabled`:

```
myButton.enabled = NO;
```

Стиль панели навигации

Сам контроллер навигации может быть отображен в одном из нескольких различных стилей. Заданный по умолчанию стиль — это стандартное серое отображение. На сегодняшний день поддерживаются три различных стиля (табл. 3.1).

Таблица 3.1

Стиль	Описание
`UIBarStyleDefault`	Стиль, заданный по умолчанию, белый текст на сером фоне

Таблица 3.1 *(окончание)*

Стиль	Описание
`UIBarStyleBlackOpaque`	Белый текст на сплошном черном фоне
`UIBarStyleBlackTranslucent`	Белый текст на прозрачном черном фоне

Стиль устанавливается с помощью свойства `barStyle`. Оно принадлежит контроллеру навигации, а не контроллеру представлений, поэтому оно остается постоянным при навигации по всем представлениям:

```
self.navigationController.navigationBar.barStyle =
  UIBarStyleBlackTranslucent;
```

Добавление сегментированного элемента управления

Элементы управления (controls) — это небольшие автономные компоненты пользовательского интерфейса, которые могут использоваться различными классами UI Kit. Они могут прикрепляться к разным типам объектов, позволяя тем самым разработчику добавлять окну дополнительную функциональность. Одним общим элементом управления, который присутствует на панелях навигации поставляемых Apple приложений, является сегментированный элемент управления (segmented control).

Во многих поставляемых приложениях вы заметите, что Apple добавила кнопки для дальнейшего разделения отображаемой информации. Например, панель навигации в приложении iTunes WiFi Store выводит кнопки **New Releases**, **What's Hot** и **Genres** наверх. Таким образом, осуществляется дальнейшее разделение музыкальных предпочтений пользователя. Сегментированные элементы управления весьма полезны в ситуациях, в которых чрезмерное количество схожих данных лучше упорядочить с помощью двух или трех кнопок.

Далее приведен пример элемента управления, снабженного двумя сегментами — "All" и "Missed":

```
UISegmentedControl *segmentedControl = [ [ UISegmentedControl alloc ]
    initWithItems: nil ];
segmentedControl.segmentedControlStyle = UISegmentedControlStyleBar;
 [ segmentedControl insertSegmentWithTitle: @"All" atIndex:
```

```
    0 animated: NO ];
 [ segmentedControl insertSegmentWithTitle: @"Missed" atIndex:
    1 animated: NO ];
```

После создания сегментированного элемента управления он может быть отображен путем присвоения его навигационному свойству `titleView` контроллера представлений. Это приведет к замене стандартного текста заголовка вашим собственным представлением:

```
self.navigationItem.titleView = segmentedControl;
```

Кроме того, вы захотите, чтобы этот класс уведомлялся всякий раз, когда пользователь выбирает новый сегмент, чтобы он мог сделать соответствующие изменения и отобразить новую информацию. Для этого воспользуйтесь методом `addTarget` класса `UIControl`, чтобы задать метод, когда значение элемента управления поменяется:

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

В этом примере селектор `controllerPressed` задан как метод, который должен уведомляться в целевом `self`. Запрограммируйте эту процедуру в вашем целевом классе, чтобы обрабатывать изменения значений:

```
- (void) controllerPressed:(id)sender {
    int selectedIndex = [ segmentedControl selectedSegmentIndex ];

    /* Дополнительный код для обработки изменения значения */
}
```

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

```
- (void) controllerPressed:(id)sender {
    int selectedSegment = segmentedControl.selectedSegmentIndex;
    NSLog(@"Segment %d selected\n", selectedSegment);
}
```

Более подробно цепочка событий класса `UIControl` и сегментированный элемент управления рассматриваются в *главе 10*.

Добавление панели инструментов

Панель в навигационном стиле (свойство — стиль) может размещать целый ряд объектов различных типов. Только что вы узнали, как прикрепить сегментированный элемент управления в качестве заглавного представления панели навигации, чтобы предоставить пользователю множество подкатегорий. Еще один распространенный компонент, который можно встретить на панели навигации, — это объект `UIToolbar`. Панели инструментов могут размещать любое задаваемое вами множество кнопок, включая стандартные системные кнопки, например, **Bookmarks** и **Search**. Многие поставляемые с iPhone приложения, например Safari и Mail, используют панели инструментов, чтобы расширить функциональность, предоставляемую панелью навигации.

Прежде чем панель инструментов сможет быть отображена, вы должны создать кнопки, которые собираетесь на нее поместить. Каждую кнопку вы добавляете в массив, созданный с помощью Cocoa-класса `NSMutableArray`:

```
NSMutableArray *buttons = [ [ NSMutableArray alloc ] init ];
```

Текстовые кнопки и кнопки с изображением

Самые распространенные типы кнопок — те, которые представлены изображениями или текстом. Оба типа кнопок создаются как объекты `UIBarButtonItem`, но для каждой используется свой инициализатор. Кнопки с изображениями инициализируются с помощью метода `initWithImage`, а стандартные текстовые кнопки — с помощью метода `initWithTitle`:

```
UIBarButtonItem *buttonImage = [[ UIBarButtonItem alloc ] initWithImage:
    [ UIImage imageNamed: @"button.png" ]
    style: UIBarButtonItemStylePlain
    target: self
    action: @selector(mySelector:)
];

UIBarButtonItem *buttonText = [[ UIBarButtonItem alloc ] initWithTitle:
    @"Button"
    style: UIBarButtonItemStyleBordered
    target: self
    action: @selector(mySelector:)
];
```

Системные кнопки

Помимо текстовых кнопок и кнопок с изображениями для создания стандартизованных предопределенных кнопок, которые вы встречаете во многих приложениях, существует небольшая библиотека системных кнопок (system buttons). Системные кнопки также создаются как объекты `UIBarButtonItem` с помощью метода `initWithBarButtonSystemItem` класса. Далее приведен соответствующий пример:

```
UIBarButtonItem *myBookmarks = [[ UIBarButtonItem alloc ]
    initWithBarButtonSystemItem: UIBarButtonSystemItemBookmarks
    target: self
    action: @selector(mySelector:)
];
```

На текущий момент поддерживаются и находятся в файле заголовков UIBarButtonItem.h следующие системные кнопки (табл. 3.2).

Таблица 3.2

Идентификатор кнопки	**Описание**
`UIBarButtonSystemItemDone`	Синяя текстовая кнопка **Done**
`UIBarButtonSystemItemCancel`	Текстовая кнопка **Cancel**
`UIBarButtonSystemItemEdit`	Текстовая кнопка **Edit**
`UIBarButtonSystemItemSave`	Синяя текстовая кнопка **Save**
`UIBarButtonSystemItemAdd`	Кнопка с изображением знака "плюс" (+)
`UIBarButtonSystemItem-FlexibleSpace`	Пустое пространство
`UIBarButtonSystemItem-FixedSpace`	Пустой разделитель
`UIBarButtonSystemItemCompose`	Кнопка с изображением ручки и бумаги
`UIBarButtonSystemItemReply`	Кнопка с изображением стрелки ответа (reply)
`UIBarButtonSystemItemAction`	Кнопка с изображением стрелки действия (action)
`UIBarButtonSystemItem-Organize`	Кнопка с изображением папки со стрелкой вниз

Таблица 3.2 *(окончание)*

Идентификатор кнопки	Описание
`UIBarButtonSystemItem-Bookmarks`	Кнопка с изображением значка избранного (bookmarks)
`UIBarButtonSystemItemSearch`	Кнопка с изображением значка прожектора (spotlight)
`UIBarButtonSystemItemRefresh`	Кнопка с изображением круговой стрелки обновления
`UIBarButtonSystemItemStop`	Кнопка с изображением знака остановки (×)
`UIBarButtonSystemItemCamera`	Кнопка с изображением камеры
`UIBarButtonSystemItemTrash`	Кнопка с изображением мусорной корзины
`UIBarButtonSystemItemPlay`	Кнопка с изображением значка воспроизведения
`UIBarButtonSystemItemPause`	Кнопка с изображением значка приостановки
`UIBarButtonSystemItemRewind`	Кнопка с изображением значка перемотки назад
`UIBarButtonSystemItem-FastForward`	Кнопка с изображением значка перемотки вперед

Собственные кнопки представлений

Как и панели навигации, кнопки могут быть представлены как собственные классы представлений, которые позволяют вам отображать в виде кнопки объект представления любого другого типа:

```
UIBarButtonItem *customButton = [ [ UIBarButtonItem alloc ]
    initWithCustomView: myView ];
```

Создание панели инструментов

Каждую кнопку, которую вы хотите отобразить, добавьте в созданный вами массив `buttons`:

```
[ buttons addObject: buttonImage ];
[ buttons addObject: buttonText ];
[ buttons addObject: myBookmarks ];
```

Затем создайте объект `UIToolbar` и задайте ваш массив кнопок как список элементов панели инструментов:

```
UIToolbar *toolbar = [ [ UIToolbar alloc ] init ];
[ toolbar setItems: buttons animated: YES ];
```

Наконец, замените заголовочное представление панели навигации вашей только что созданной панели инструментов точно так же, как вы делали это с сегментированным элементом управления:

```
self.navigationItem.titleView = toolbar;
```

При отображении панели навигации панель инструментов появится по центру.

Изменение размеров

Для добавляемых на панели инструментов кнопок используется размер по умолчанию. Если вы хотите более четко подогнать размер панели инструментов под размер панели навигации, воспользуйтесь методом `sizeToFit`:

```
[ toolbar sizeToFit ];
```

Стиль панели инструментов

Как и в случае со многими объектами на базе представления, `UIToolbar` включает свойство стиля `barStyle`. Оно может использоваться для выбора стиля, который соответствовал бы стилю, установленному для панели навигации:

```
toolbar.barStyle = UIBarStyleDefault;
```

Для панели инструментов вы можете выбрать один из трех стандартных стилей, используемых большинством объектов панелей других типов (табл. 3.3).

Таблица 3.3

Стиль	Описание
`UIBarStyleDefault`	Стиль по умолчанию; фон серого цвета, текст белого цвета
`UIBarStyleBlackOpaque`	Сплошной фон черного цвета, текст белого цвета
`UIBarStyleBlackTranslucent`	Прозрачный черный цвет фона, текст белого цвета

Страничная навигация: PageDemo

В этом примере вы создадите контроллер навигации поверх существующего контроллера представлений. Свойства навигации корневого контроллера представлений добавляют кнопку **Credits** так, чтобы при ее нажатии в стек контроллера навигации проталкивался другой контроллер представлений. Когда заглавные титры отображены, нажатие кнопки **Back** выталкивает контроллер представлений из стека, переходя обратно к корневому представлению. Кроме того, к корневому представлению добавится сегментированный элемент управления, позволяющий пользователю выбирать между отображением текста о кроликах и отображением текста о пони. Внешний вид этих элементов управления представлен на рис. 3.2.

Рис. 3.2. Пример PageDemo

Вы можете скомпилировать это приложение, показанное в листингах 3.17—3.21, с помощью SDK, создав проект навигационного приложения PageDemo. Если вы хотите увидеть, как создавать все эти объекты с нуля, то удалите код Interface Builder.

Листинг 3.17. Прототипы делегата приложения PageDemo (PageDemoAppDelegate.h)

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

@interface PageDemoAppDelegate : NSObject <UIApplicationDelegate> {
    UIWindow *window;
    RootViewController *viewController;
    CreditsViewController *creditsViewController;
    UINavigationController *navigationController;
}
```

```
@property (nonatomic, retain) IBOutlet UIWindow *window;
@property (nonatomic, retain) IBOutlet RootViewController
   *viewController;
@property (nonatomic, retain) IBOutlet CreditsViewController
   *creditsViewController;
@property (nonatomic, retain) IBOutlet UINavigationController
   *navigationController;

@end
```

Листинг 3.18. Делегат приложения PageDemo (PageDemoAppDelegate.m)

```
#import "PageDemoAppDelegate.h"
#import "RootViewController.h"

@implementation PageDemoAppDelegate

@synthesize window;
@synthesize viewController;
@synthesize navigationController;
@synthesize creditsViewController;

- (void)applicationDidFinishLaunching:(UIApplication *)application {

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

  viewController = [[RootViewController alloc] initWithAppDelegate:
   self];

  creditsViewController = [ [ CreditsViewController alloc ]
      initWithAppDelegate: self
  ];

  navigationController = [ [ UINavigationController alloc ]
      initWithRootViewController: viewController
  ];
```

```
  [ window addSubview: [ navigationController view ] ];
  [ window makeKeyAndVisible ];
}

- (void)dealloc {
    [ navigationController release ];
    [ viewController release ];
    [ creditsViewController release ];
    [ window release ];
    [ super dealloc ];
}

- (void)credits {
    [ navigationController pushViewController: creditsViewController
        animated: YES
    ];
}

- (void)back {
    [ navigationController popViewControllerAnimated: YES ];
}

@end
```

Листинг 3.19. Прототипы контроллера представлений PageDemo (RootViewController.h)

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

@interface RootViewController : UIViewController {
    UITextView *textView;
    UIBarButtonItem *credits;
    UISegmentedControl *segmentedControl;
    UINavigationController *navigationController;
    int page;
```

```
}
- (void)setPage;
- (id)initWithAppDelegate:(id)appDelegate;

@end

@interface CreditsViewController : UIViewController {
    UITextView *textView;
    UINavigationController *navigationController;
}
- (id)initWithAppDelegate:(id)appDelegate;

@end
```

Листинг 3.20. Контроллеры представлений PageDemo (RootViewController.m)

```
#import "RootViewController.h"
#import "PageDemoAppDelegate.h"

@implementation RootViewController

- (id)initWithAppDelegate:(id)appDelegate {
  self = [ super init ];
  if (self != nil) {
    /* Инициализируем навигационные кнопки */
    credits = [ [ [ UIBarButtonItem alloc ]
                initWithTitle:@"Credits"
                style: UIBarButtonItemStylePlain
                target: appDelegate
                action:@selector(credits) ]
                autorelease ];
    self.navigationItem.rightBarButtonItem = credits;

    segmentedControl = [ [ UISegmentedControl alloc ] initWithItems: nil
];
    segmentedControl.segmentedControlStyle = UISegmentedControlStyleBar;
```

```
    [ segmentedControl insertSegmentWithTitle: @"Bunnies" atIndex: 0
            animated: NO
    ];
    [ segmentedControl insertSegmentWithTitle: @"Ponies" atIndex: 1
            animated: NO
    ];

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

    self.navigationItem.titleView = segmentedControl;
        segmentedControl.selectedSegmentIndex = 0;
  }
  return self;
}

- (void)controlPressed:(id) sender {
    [ self setPage ];
}

- (void)setPage {
  int index = segmentedControl.selectedSegmentIndex;

  if (index == 0) {
      textView.text = @"OMG Bunnies!";
  } else {
      textView.text = @"OMG Ponies";
  }
}

- (void)loadView {
  CGRect bounds = [ [ UIScreen mainScreen ] applicationFrame ];
  [ super loadView ];

  textView = [ [ UITextView alloc ] initWithFrame: bounds ];
  textView.editable = NO;
```

```
  [ self setPage ];
  self.view = textView;
}

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

@end

@implementation CreditsViewController

- (id)initWithAppDelegate:(id)appDelegate {

  self = [ super init ];
  if (self != nil) {

    /* Инициализируем навигационные кнопки */
    UIBarButtonItem *back = [ [ [ UIBarButtonItem alloc ]
                  initWithTitle:@"Back"
                  style: UIBarButtonItemStylePlain
                  target: appDelegate
                  action: @selector(back) ]
                autorelease ];
    self.navigationItem.backBarButtonItem = back;
  }
  return self;
}

- (void)loadView {
    [ super loadView ];

    textView = [ [ UITextView alloc ] initWithFrame: [
        [ UIScreen mainScreen ] applicationFrame ] ];
```

```
    textView.editable = NO;
    textView.text = @"iPhone SDK Application Development\n"
                    "Copyright (c) 2008, O'Reilly Media.";

    self.view = textView;
}

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

@end
```

Листинг 3.21. Функция `main` для PageDemo (main.m)

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

int main(int argc, char *argv[]) {
  NSAutoreleasePool * pool = [[NSAutoreleasePool alloc] init];
  int retVal = UIApplicationMain(argc, argv, nil,
@"PageDemoAppDelegate");
  [pool release];
  return retVal;
}
```

Как это работает

Пример PageDemo содержит все, что вы до этого момента видели, и добавляет контроллер панели навигации для существующего контроллера представлений, чтобы управлять навигацией между ним и другим контроллером представлений:

1. Приложение порождается, и вызывается программная функция `main`, которая вызывает метод `applicationDidFinishLaunching` класса `PageDemoAppDelegate`.
2. Делегат приложения создает экземпляр контроллера корневого представления и контроллер представления титров. Для инициализации контрол-

лера представлений и передачи указателя объекту делегата приложения, который будет обрабатывать действия для каждой кнопки, используется собственный метод инициализации `initWithAppDelegate`. Также добавляется сегментированный элемент управления.

3. После того как контроллеры представлений завершат инициализацию, продолжит свою работу делегат приложения, порождая объект контроллера навигации, прикрепляя контроллер представлений в качестве его корня. Затем контроллер панели навигации прикрепляется к окну как подчиненное представление, в котором он отображает себя и контроллер представлений, который он обертывает.

4. Когда пользователь на панели навигации касается кнопки **Credits**, действие этой кнопки вызывает метод `credits` в классе делегата приложения. Тем самым объект `creditsViewController` проталкивается в контроллер навигации, автоматически переводя экран в новое представление. Когда пользователь нажимает кнопку **Back**, действие этой кнопки вызывает метод `back` в классе делегата приложения. Тем самым этот контроллер выталкивается из стека, возвращая экран iPhone к предыдущему контроллеру представлений.

5. Когда пользователь касается кнопки на сегментированном элементе управления, метод селектора вызывает метод `setPage`. Текст, присвоенный текстовому представлению, зависит от того, что выбрал пользователь: "bunnies" или "ponies" в сегментированном элементе управления.

Для дальнейшего изучения

Попробуйте немного протестировать код из этого раздела и из предыдущих примеров.

- Попробуйте изменить этот код и добавить в сегментированный элемент управления третий сегмент, "Hax". При выборе пользователем этого сегмента должен отображаться текст "OMG Hax!".

- Возьмите код `UITextView` из предыдущих примеров и добавьте две кнопки: одну для HTML, а другую — для текста. Нажатие на каждую кнопку должно изменять текстовое представление для отображения файла в соответствующем формате.

- Замените сегментированный элемент управления панелью инструментов. Добавьте системную кнопку `UIBarButtonSystemItemAdd`, чтобы при касании ее текстовое представление становилось редактируемым, и кнопку

`UIBarButtonSystemItemDone`, чтобы при касании ее текстовое представление снова переходило в режим только для чтения.

- Добавьте третий контроллер представлений, отображающий HTML-представление текста. Протолкните его в ваш стек навигации и добавьте соответствующие кнопки для обратной навигации.
- В заголовочных файлах вашего SDK проверьте наличие следующих прототипов: UINavigationController.h, UINavigationBar.h и UIBarButtonItem.h. Вы найдете их в папке /Developer/Platforms/iPhoneOS.platform внутри каталога Headers библиотеки UI Kit.

Анимации переходов

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, могут предоставлять возможность плавных переходов, предлагаемую UI Kit. Переходы (transitions) являются анимационными объектами, прикрепленными к существующему представлению так, что при изменении представления выполняется анимация.

К счастью, большая часть работы по реализации переходов автоматически выполняется за вас при использовании контроллеров навигации. Тем не менее, могут существовать ситуации, когда может оказаться полезным прямое применение перехода. Управлением анимацией переходов занимается библиотека, о которой мы еще не упоминали, — Quartz Core. Эта библиотека предоставляет целый ряд анимаций и преобразований, используемых для создания множества различных эффектов. Для создания анимации перехода и применения ее к вашему представлению воспользуйтесь процессором анимации Quartz Core. Для перехвата всей обработки графики во время перехода Quartz Core порождает новый поток. Разработчику требуется только добавить желаемый переход для расширения возможностей существующего приложения.

Чтобы реализовать переходы в вашем приложении, вам потребуется добавить библиотеку Quartz Core в ваш проект. Щелкните правой кнопкой мыши по папке Frameworks в вашем проекте, а затем в появившемся контекстном меню выберите команду **Add Framework**. Перейдите к папке QuartzCore.framework, а затем нажмите кнопку **Add**.

Создание перехода

Переходы создаются как объекты `CATransition`, являющиеся объектами, принадлежащими набору функцией Core Animation из Quartz Core. Анимация содержит такие свойства, как функция распределения во времени, тип анимации и длительность. Чтобы создать анимацию, вызовите метод `animation` класса. Вот соответствующий пример:

```
#import <QuartzCore/CAAnimation.h>

CATransition *myTransition = [ CATransition animation ];
```

Функция распределения во времени

Функция распределения во времени (timing) определяет, насколько плавно выполняется переход от начала до конца. Одна и та же анимация может выполняться с разными скоростями. Например, анимация переворачивания страниц может начинаться медленно и увеличивать скорость в конце или же наоборот. Для ваших анимаций предоставляются следующие функции распределения во времени (табл. 3.4).

Таблица 3.4

Распределение во времени	Описание
`UIViewAnimationCurveEaseInOut`	Медленно в начале и в конце анимации
`UIViewAnimationCurveEaseIn`	Медленно в начале, затем ускорение
`UIViewAnimationCurveEaseOut`	Анимация замедляется в конце
`UIViewAnimationCurveLinear`	Одинаковая скорость в течение всей анимации

Установите функцию распределения во времени, определив значение свойства `timingFunction`:

```
myTransition.timingFunction = UIViewAnimationCurveEaseInOut;
```

Типы анимации

Тип анимации (animation type) определяет вид визуализируемой анимации. Каждая анимация имеет тип и подтип. Тип анимации определяет общее поведение перехода, а подтип — такие нюансы, как направление перехода.

Apple предоставляет в SDK разработчикам ограниченный доступ к существующим анимациям, поэтому вы не сможете в ваш код добавить все, что видите в поставляемых Apple приложениях. SDK поддерживает следующие типы анимации (табл. 3.5).

Таблица 3.5

Тип анимации	**Описание**
`kCATransitionFade`	Постепенное изменение одного представления на следующее
`kCATransitionMoveIn`	Перемещение нового представления поверх старого
`kCATransitionPush`	Выталкивание старого представления и помещение нового
`kCATransitionReveal`	Перемещение старого представления или открытие нового

Поддерживаются следующие подтипы анимации (табл. 3.6).

Таблица 3.6

Подтип анимации	**Описание**
`kCATransitionFromRight`	Новое представление соскальзывает справа
`kCATransitionFromLeft`	Новое представление соскальзывает слева
`kCATransitionFromTop`	Новое представление соскальзывает сверху
`kCATransitionFromBottom`	Новое представление соскальзывает снизу

Тип и подтип можно задать путем присвоения значений свойствам `type` и `subtype`:

```
myTransition.type = kCATransitionPush;
myTransition.subtype = kCATransitionFromLeft;
```

Длительность

По умолчанию для каждой анимации установлена стандартная длительность, при которой она выполняется на нормальной скорости. Как правило, для большинства анимаций это 0,3 секунды. Если вы хотите ускорить анимацию

или замедлить ее, можете вручную установить длительность, задав значение свойства `duration`:

```
myTransition.duration = 0.3;
```

Прикрепление перехода

Чтобы выполнить переход, добавьте анимацию к тому уровню, к которому ваше анимированное представление прикреплено. Например, если вы осуществляете переход между двумя контроллерами представлений, то добавьте анимацию к уровню окна:

```
[ [ self.view.superview.layer ] addAnimation: myTransition forKey: nil ];
```

Если вы осуществляете переход от одного дочернего представления к другому в рамках одного контроллера представлений, то добавьте анимацию к уровню контроллера представлений. Иначе вы можете воспользоваться объектом представления внутри контроллера представлений и управлять вашими дочерними представлениями как подуровнями главного представления:

```
[ self.view.layer addAnimation: myTransition forKey: nil ];
[ self.view addSubview: newView ];
[ oldView removeFromSuperview ];
```

Если вы используете контроллер навигации, как в примере PageDemo, то для осуществления перехода между двумя контроллерами представлений вы можете добавить анимацию к уровню представления контроллера навигации:

```
[ navigationController.view.layer addAnimation:
    myTransition forKey: nil ];
```

Как только произойдет переключение между представлениями, автоматически начнет свое выполнение уровень анимации, отображая тем самым переход между этими двумя представлениями.

Переходы с переворачиванием страниц: FlipDemo

Этот пример работает аналогично уже рассмотренному ранее примеру PageDemo. В данном примере вы создадите контроллер навигации поверх существующего контроллера представлений. Этот пример отображает элемент левой и правой панели навигации, позволяя пользователю перелистывать де-

сять страниц текста в рамках единственного контроллера представлений. Когда это происходит, создается новая анимация и добавляется к родительскому представлению для отображения данного перехода пользователю. Также добавляется общий объект `UIView`, выступающий в роли родительского представления в рамках контроллера представлений, поэтому при инициализации класса вместо `self.view`, указывающего на отдельное текстовое представление, создаются все десять текстовых представлений. Затем каждый раз при осуществлении перехода добавляются и удаляются соответствующие страницы как подуровни общего представления.

Вы можете скомпилировать это приложение, представленное в листингах 3.22—3.26, с помощью SDK, создав проект навигационного приложения FlipDemo. Если вы хотите увидеть, как создавать все эти объекты с нуля, то удалите код Interface Builder.

> ***Внимание!***
>
> Перед попыткой создать проект убедитесь в том, что вы добавили к нему библиотеку Quartz Core.

Листинг 3.22. Прототипы делегата приложения FlipDemo (FlipDemoAppDelegate.h)

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

@interface FlipDemoAppDelegate : NSObject <UIApplicationDelegate> {
    UIWindow *window;
    RootViewController *viewController;
    UINavigationController *navigationController;
}

@property (nonatomic, retain) IBOutlet UIWindow *window;
@property (nonatomic, retain) IBOutlet RootViewController
  *viewController;
@property (nonatomic, retain) IBOutlet UINavigationController
  *navigationController;

@end
```

Листинг 3.23. Делегат приложения FlipDemo (FlipDemoAppDelegate.m)

```
#import "FlipDemoAppDelegate.h"
#import "RootViewController.h"

@implementation FlipDemoAppDelegate

@synthesize window;
@synthesize viewController;
@synthesize navigationController;

- (void)applicationDidFinishLaunching:(UIApplication *)application {
    window = [ [ UIWindow alloc ] initWithFrame:
      [ [ UIScreen mainScreen ] bounds ] ];
    viewController = [ [ RootViewController alloc ] init ];
    navigationController = [ [ UINavigationController alloc ]
        initWithRootViewController: viewController ];

    [ window addSubview: [ navigationController view ] ];
    [ window makeKeyAndVisible ];
}

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

@end
```

Листинг 3.24. Прототип контроллера представлений FlipDemo (RootViewController.h)

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

@interface RootViewController : UIViewController {
    UITextView *textView[10];
```

```
    UIView *view;
    UIBarButtonItem *prev, *next;
    int page, lastViewed;
}
- (void)setPage;

@end
```

Листинг 3.25. Контроллер представлений FlipDemo (RootViewController.m)

```
#import <QuartzCore/CAAnimation.h>
#import "RootViewController.h"
#import "FlipDemoAppDelegate.h"

@implementation RootViewController

- (id)init {
  self = [ super init ];
  if (self != nil) {

    /* Страница по умолчанию */
    page = lastViewed = 5;

    /* Инициализируем навигационные кнопки */
    prev = [ [ [ UIBarButtonItem alloc ]
                 initWithTitle:@"Prev"
                 style: UIBarButtonItemStylePlain
                 target: self
                 action:@selector(prevpage) ]
               autorelease ];
    self.navigationItem.leftBarButtonItem = prev;
    next = [ [ [ UIBarButtonItem alloc ]
                 initWithTitle:@"Next"
                 style: UIBarButtonItemStylePlain
                 target: self
                 action:@selector(nextpage) ]
               autorelease ];
```

```
    self.navigationItem.rightBarButtonItem = next;
  }
  return self;
}

- (void)controlPressed:(id) sender {
    [ self setPage ];
}

- (void)setPage {
    /* Создаем новую анимацию */
    CATransition *myTransition = [ CATransition animation ];
    myTransition.timingFunction = UIViewAnimationCurveEaseInOut;
    myTransition.type = kCATransitionPush;
    if (page > lastViewed) {
        myTransition.subtype = kCATransitionFromRight;
    } else {
        myTransition.subtype = kCATransitionFromLeft;
    }

    /* Добавляем анимацию к верхнему уровню представления */

    [ self.view.layer addAnimation: myTransition forKey: nil ];
    [ self.view insertSubview: textView[page-1] atIndex: 0 ];
    [ textView[lastViewed-1] removeFromSuperview ];

    lastViewed = page;

    if (page == 1)
        prev.enabled = NO;
    else
        prev.enabled = YES;

    if (page == 10)
        next.enabled = NO;
    else
```

```
        next.enabled = YES;
}

- (void)prevpage {
    page--;
    [ self setPage ];
}

- (void)nextpage {
    page++;
     [ self setPage ];
}

- (void)loadView {
    CGRect bounds = [ [ UIScreen mainScreen ] applicationFrame ];
    [ super loadView ];

    view = [ [ UIView alloc ] initWithFrame: bounds ];
    bounds.origin.y = 0;

    for(int i = 0;i < 10; i++) {
      textView[i] = [ [ UITextView alloc ] initWithFrame: bounds ];
      textView[i].editable = NO;
      textView[i].text = [ NSString stringWithFormat: @"Page %d", i+1 ];
    }

    self.view = view;
    [ self.view addSubview: textView[4] ];
}

- (void)dealloc {
    for(int i = 0; i < 10; i++) {
        [ textView[i] dealloc ];
    }
    [ next dealloc ];
```

```
    [ prev dealloc ];
    [ super dealloc ];
}

@end
```

Листинг 3.21. Функция `main` для FlipDemo (main.m)

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

int main(int argc, char *argv[]) {
    NSAutoreleasePool * pool = [[NSAutoreleasePool alloc] init];
    int retVal = UIApplicationMain(argc, argv, nil,
                                   @"FlipDemoAppDelegate");
    [pool release];
    return retVal;
}
```

Как это работает

Пример FlipDemo содержит все, что вы до этого момента видели, и добавляет анимацию к родительскому представлению:

1. Приложение порождается, и вызывается программная функция `main`, которая вызывает метод `applicationDidFinishLaunching` класса `FlipDemoAppDelegate`.
2. Делегат приложения создает экземпляр контроллера представления. Метод `init` контроллера представлений подменяется для определения свойств панели навигации, используемых при отображении, а именно двух кнопок панели навигации и сегментированного элемента управления.
3. Контроллер представлений создает десять различных объектов `UITextView`, каждый из которых содержит свой текст. Также создается общий объект `UIView` и назначается в качестве ключевого представления для контроллера. После завершения инициализации контроллера представлений продолжает свою работу делегат приложения, порождая объект контроллера навигации, используя для этого контроллер представлений в качестве корневого контроллера. Затем к окну прикрепляется контроллер панели навигации в качестве подчиненного представления, в котором он отображает себя и контроллер представлений, который он обертывает.

4. Когда пользователь на панели навигации касается кнопки, вызывается действие этой кнопки. Селектор устанавливает номер страницы, а затем уведомляет метод `setPage`. Этот метод создает анимацию и прикрепляет ее к ключевому представлению контроллера. Анимация переворачивает страницы влево или вправо в соответствии с направлением переворачивания страниц.
5. Представление следующей (или предыдущей) страницы добавляется к ключевому представлению, а старые страницы удаляются. Когда это происходит, анимация выполняется автоматически, отображая плавное перемещение страниц.

Для дальнейшего изучения

Прежде чем продолжить, изучите немного глубже анимацию переходов.

- Попробуйте создать другие типы переходов, используя типы переходов, упомянутые ранее в этом разделе.
- Возьмите код `UITextView` из предыдущих примеров и добавьте две кнопки: одну для HTML, а другую — для текста. Нажатие на каждую кнопку должно изменять текстовое представление для отображения файла в соответствующем формате. При переключении между этими двумя представлениями должен происходить переход сверху вниз и наоборот.
- В заголовочных файлах вашего SDK проверьте наличие следующих прототипов: CAAnimation.h, CAMediaTimingFunction.h и CALayer.h. Вы найдете их в папке /Developer/Platforms/iPhoneOS.platform внутри каталога Headers библиотеки QuartzCore.

Листы действий и предупреждения

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 нажимать не те кнопки. Если такое происходит, то хорошо продуманное 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 всплывает модальный лист предупреждений (alert sheet), 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. На самом деле iPhone действительно поддерживает всплывающие окна в форме предупреждений. Как правило, они передают информацию, хотя иногда запрашивают еще и пользовательский ввод.

Предупреждения

Представления предупреждений (alert views) являются простейшими предупреждениями, поддерживаемыми iPhone. Представление предупреждения, реализованное классом `UIAlertView`, — это небольшое всплывающее окно, появляющееся поверх окна приложения. Пока предупреждение не будет обработано пользователем, доступ к самому приложению остается заблокированным:

```
UIAlertView *myAlert = [ [ UIAlertView alloc ]
    initWithTitle:@"Alert"
    message:@"This is an alert."
    delegate:self
    cancelButtonTitle: nil
    otherButtonTitles: @"OK", nil];
```

Когда пользователь в окне предупреждения нажимает кнопку, представление предупреждения уведомляет своего делегата, вызывая метод `clickedButtonAtIndex`. Этот метод принадлежит протоколу `UIAlertViewDelegate`, который обязан реализовывать класс-делегат. Чтобы задать класс-делегат, который будет получать эти уведомления, задайте значение свойства `delegate` представления предупреждения:

```
myAlert.delegate = self;
```

Затем вам необходимо добавить этот специальный метод к вашему классу-делегату для обработки нажатий кнопок. Вот соответствующий пример:

```
- (void)alertView:(UIAlertView *)alertView
     clickedButtonAtIndex:(NSInteger)buttonIndex
{
    NSLog(@"Button %d pressed", buttonIndex);
    [ alertView release ];
}
```

Метод `clickedButtonAtIndex` предоставляется с указателем на объект `UIAlertView` и индекс кнопки. Указатель можно сравнивать с существующими

объектами для определения того, в каком представлении предупреждения нажата кнопка, что позволяет использовать единственного делегата для обработки множественных предупреждений. Параметр `buttonIndex` предоставляет номер кнопки (нумерация начинается с нуля), соответствующий кнопке в предупреждении.

Когда вы будете готовы отобразить предупреждение, воспользуйтесь методом `show` класса `UIAlertView`:

```
[ myAlert show ];
```

Листы действий

Листы действий (action sheets) — это расширенные типы предупреждений, которые появляются в виде "выскальзывающих" (slide-up) страниц поверх существующего представления. Одно представление может хранить множество различных листов действий и обрабатывать их, используя те же возможности делегата, что и представления предупреждений. Простейший лист действий состоит из заголовка, сообщения и тех вариантов выбора, которые должны быть представлены пользователю. Он порождается точно так же, как и представление предупреждения:

```
UIActionSheet *mySheet = [ [ UIActionSheet alloc ]
    initWithTitle: @"Please Select"
    delegate: self
    cancelButtonTitle: @"Cancel"
    destructiveButtonTitle: @"Delete"
    otherButtonTitles: @"Move", @"Rename", nil ];
```

Когда пользователь нажимает кнопку, делегат уведомляется путем вызова метода `clickedButtonAtIndex`. В случае листов действий этот метод принадлежит протоколу `UIActionSheetDelegate`:

```
- (void)actionSheet:(UIActionSheet *)actionSheet
    clickedButtonAtIndex:(NSInteger)buttonIndex
{
    /* Здесь обработайте лист действий */
}
```

Листы действий поддерживают тип кнопок, известных под названием *кнопки уничтожения* (destructive buttons). Кнопка уничтожения в листе действий

предназначена для того, чтобы обозначить действие полного уничтожения, как например, удаление сообщения или очистка журнала. Если вы задаете кнопку уничтожения, то она выделяется красным цветом, предупреждая тем самым пользователя о своем потенциально опасном действии:

```
mySheet.destructiveButtonIndex = 1;
```

Как панели навигации и панели инструментов, листы действий могут поддерживать один из следующих трех стилей (табл. 3.7).

Таблица 3.7

Стиль	**Описание**
`UIActionSheetStyleDefault`	Стиль по умолчанию, фон серого цвета, текст белого цвета
`UIActionSheetStyleBlack-Translucent`	Прозрачный черный цвет фона, текст белого цвета
`UIActionSheetStyleBlackOpaque`	Фон сплошного черного цвета, текст белого цвета

Задать стиль можно путем присвоения соответствующего значения свойству `actionSheetStyle` листа действий:

```
mySheet.actionSheetStyle = UIActionSheetStyleDefault;
```

Для отображения листа действий можно использовать любой из трех методов. Если лист действий будет взят из представления, то воспользуйтесь методом `showInView`, чтобы лист выскользнул с нижней части представления:

```
[ mySheet showInView: self ];
```

Чтобы выровнять лист действий по краю панели инструментов или панели вкладок, воспользуйтесь методом `showFromToolBar` или методом `showFromTabBar`:

```
[ mySheet showFromToolBar: toolbar ];
[ mySheet showFromTabBar: tabbar ];
```

Отмена листа действий

После обработки нажатия кнопки лист действий должен исчезнуть с экрана, если только, конечно же, приложению не требуется, чтобы пользователь на-

жал еще и другие кнопки. Чтобы убрать лист с экрана, воспользуйтесь методом `dismiss`:

```
[ mySheet dismissWithClickedButtonIndex: 1 animated: YES ];
```

Конец света (с подтверждением): EndWorld

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 EndWorld.app, которое президент может запустить в любой момент времени, чтобы пустить в ход ядерное оружие, которое приведет к концу света (или, по крайней мере, начнется передел мира). Однако существует проблема, связанная с тем, что президент может случайно нажимать ее, думая при этом, что он просто общается в социальных сетях. Представьте, что с вами заключили контракт на добавление к этому приложению подтверждения на случай, если президент на самом деле не хотел, чтобы наступил конец света (рис. 3.3).

Рис. 3.3. Пример EndWorld

Приложение EndWorld отображает панель навигации с кнопкой **End World** и удобное текстовое окно для ввода последних слов. Когда президент нажмет

кнопку конца света, приложение сначала запросит подтверждение посредством представления предупреждения.

Вы можете скомпилировать это приложение, показанное в листингах 3.27—3.31, с помощью SDK, создав проект навигационного приложения EndWorld. Если вы хотите увидеть, как создавать все эти объекты с нуля, то удалите код Interface Builder.

Листинг 3.27. Прототипы делегата приложения EndWorld (EndWorldAppDelegate.h)

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

@interface EndWorldAppDelegate : NSObject <UIApplicationDelegate> {
    UIWindow *window;
    RootViewController *viewController;
    UINavigationController *navigationController;
}

@property (nonatomic, retain) IBOutlet UIWindow *window;
@property (nonatomic, retain) IBOutlet RootViewController
  *viewController;
@property (nonatomic, retain) IBOutlet UINavigationController
  *navigationController;

@end
```

Листинг 3.28. Делегат приложения EndWorld (EndWorldAppDelegate.m)

```
#import "EndWorldAppDelegate.h"
#import "RootViewController.h"

@implementation EndWorldAppDelegate

@synthesize window;
@synthesize viewController;
@synthesize navigationController;
```

```
- (void)applicationDidFinishLaunching:(UIApplication *)application {

    window = [ [ UIWindow alloc ] initWithFrame:
        [ [ UIScreen mainScreen ] bounds ] ];
    viewController = [ [ RootViewController alloc ] init ];
    navigationController = [ [ UINavigationController alloc ]
        initWithRootViewController: viewController ];

    [ window addSubview: [ navigationController view ] ];
    [ window makeKeyAndVisible ];
}

- (void)dealloc {
    [ navigationController release ];
    [ viewController release ];
    [ window release ];
    [ super dealloc ];
}

@end
```

Листинг 3.29. Прототип контроллера представлений EndWorld (RootViewController.h)

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

@interface RootViewController : UIViewController {
    UITextView *textView;
    UIBarButtonItem *endWorld;
    UIAlertView *endWorldAlert;
}
- (id)init;
- (void) endWorld;

@end
```

Листинг 3.30. Контроллер представлений EndWorld (RootViewController.m)

```
#import "RootViewController.h"
#import "EndWorldAppDelegate.h"

@implementation RootViewController

- (id)init {
    self = [ super init ];
    if (self != nil) {
      /* Инициализируем навигационные кнопки */
      endWorld = [ [ [ UIBarButtonItem alloc ]
                  initWithTitle:@"End World"
                  style: UIBarButtonItemStyleDone
                  target: self
                  action:@selector(endWorld) ]
                autorelease ];
      self.navigationItem.rightBarButtonItem = endWorld;
    }
    return self;
}

- (void)loadView {
    CGRect bounds = [ [ UIScreen mainScreen ] applicationFrame ];
    [ super loadView ];

    textView = [ [ UITextView alloc ] initWithFrame: bounds ];
    textView.editable = YES;
    textView.text = @"Enter any last words here, then press End World.";

    self.view = textView;
}

- (void)endWorld {
    endWorldAlert = [ [ UIAlertView alloc ]
        initWithTitle:@"End The World"
```

```
        message:@"Warning: You are about to end the world."
        delegate:self
        cancelButtonTitle: @"My Bad"
        otherButtonTitles: @"OK", nil];

    endWorldAlert.delegate = self;
    [ endWorldAlert show ];
}

- (void)alertView:(UIAlertView *)alertView
     clickedButtonAtIndex:(NSInteger)buttonIndex
{
  if (alertView == endWorldAlert) {
    NSLog(@"Button %d pressed", buttonIndex);

    UIAlertView *myAlert = [ [ UIAlertView alloc ]
            initWithTitle:@"End The World"
            message: nil
            delegate:self
            cancelButtonTitle: nil
            otherButtonTitles: @"OK", nil ];

    if (buttonIndex == 0) {
      myAlert.message = @"Be more careful next time!";
    } else if (buttonIndex == 1) {
      myAlert.message =
         @"You must be connected to a WiFi network to end the world.";
    } else {
      myAlert.message = @"Invalid Button.";
    }

    [ myAlert show ];
  }

  [ alertView release ];
}
```

```
- (void)dealloc {
    [ textView dealloc ];
    [ endWorld dealloc ];
    [ super dealloc ];
}

@end
```

Листинг 3.31. Функция `main` для EndWorld (main.m)

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

int main(int argc, char *argv[]) {
    NSAutoreleasePool * pool = [[NSAutoreleasePool alloc] init];
    int retVal = UIApplicationMain(argc, argv, nil,
                                   @"EndWorldAppDelegate");
    [pool release];
    Action Sheets and Alerts | 93
    return retVal;
}
```

Как это работает

Пример EndWorld иллюстрирует передачу управления при использовании делегата:

1. Приложение порождается точно так же, как и все предыдущие примеры, и вызывается программная функция `main`, которая вызывает метод `applicationDidFinishLaunching` класса `EndWorldAppDelegate` и создает соответствующее окно, контроллер представлений и классы контроллера навигации.
2. Метод `init` контроллера представлений подменяется для добавления кнопки **End World** панели навигации. При касании этой кнопки уведомляется селектор данной кнопки — `endWorld`.
3. Селектор создает новый объект `UIAlertView` и представляет его пользователю. Когда пользователь нажимает одну из кнопок, уведомляется делегат предупреждения (на данный момент контроллер представлений) путем вызова метода `clickedButtonAtIndex`.

4. Метод `clickedButtonAtIndex` определяет, какая кнопка была нажата, и строит новый объект `UIAlertView` для информирования пользователя о результатах его выбора. Опа, вам необходимо быть в сети WiFi не только для того, чтобы пользоваться магазином iTunes, но и для того, чтобы покончить с миром. Кто бы мог подумать!

Для дальнейшего изучения

Чтобы получить более глубокое понимание того, как работают представления предупреждений и листы действий, потестируйте немного приложение.

- Конвертируйте представление предупреждения из этого примера в лист действий. Поэкспериментируйте с различными кнопками. Сколько кнопок поместится на экране? Сколько текста на них поместится? Каким образом лист будет менять свои размеры в соответствии с вашим выбором?
- Создайте представление предупреждения без кнопок, информирующее пользователя о том, что загружается какой-либо файл. Воспользуйтесь NSTimer для ожидания в 10 секунд, а затем отмените представление, не используя метод `buttonClicked`.
- В заголовочных файлах вашего SDK проверьте наличие следующих прототипов: UIAlert.h и UITextField.h. Вы найдете их в папке /Developer/Platforms/iPhoneOS.platform внутри каталога Headers библиотеки UI Kit.

Табличные представления и контроллеры

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. Специально для таблиц существует отдельный подкласс контроллера представлений. Класс `UITableViewController` инкапсулирует один объект `UITableView` и предоставляет средства для обработки вашей бизнес-логики, поворотов экрана, свойств панели навигации и

всех других полезных функций, которые возникают при использовании контроллеров представлений.

Создание таблицы

Таблица (table) имеет три основные составляющие: саму таблицу, разделы таблицы (или группировки) и ячейки таблицы (отдельные строки в таблице). Данные таблицы помещаются в очередь из *источника данных* (data source) таблицы. Источник данных — это объект, предоставляющий таблице информацию о том, какие данные отображать, например, имена файлов, сообщения электронной почты и т. д. Источник данных должен реализовывать протокол `UITableViewDataSource` и отвечать определенному подмножеству методов, предоставляя таблице эту информацию.

Когда вы создаете таблицы, вы предоставляете указатель на объект, выступающий в роли источника данных. Источник данных будет вызываться всякий раз при перезагрузке таблицы или прокручивании в зону видимости новых ячеек таблицы, поэтому таблица сможет получать инструкции о том, какие разделы и строки необходимо отображать, и предоставлять для них данные.

Наследование класса *UITableViewController*

Таблица, как собственный источник данных, вполне подходит для большинства специфических применений. Это позволяет классу таблицы и данным таблицы быть аккуратно обернутыми в один класс контроллера. На самом деле класс `UITableViewController` уже создан для реализации протокола `UITableViewDataSource`.

Создайте производный класс объекта `UITableViewController`. В приведенном далее примере создается подкласс `MyTableViewController`. Чтобы интегрировать табличную часть класса, подменяются методы базового класса, используемые для инициализации и разрушения объекта:

```
@interface MyTableViewController : UITableViewController
{

}
- (id)init;
- (void)dealloc;
```

Чтобы добавить часть класса, касающуюся источника данных, вам придется написать три метода, которые будут отвечать на запросы данных от привязки данных (data binding): `numberOfSectionsInTableView`, `numberOfRowsInSections` и `cellForRowAtIndexPath`. Поскольку контроллер таблицы выступает в роли источника данных, то вы будете писать эти методы в производном классе вашего контроллера.

Для создания множества уникальных групп внутри таблицы воспользуйтесь методом `numberOfSectionsInTableView`. Этот метод чаще всего используется при создании сгруппированных таблиц настроек или списков разделов. Более подробно о нем вы узнаете в *главе 10*. Стандартная таблица состоит только из одного раздела, вот ее код:

```
- (NSInteger)numberOfSectionsInTableView:(UITableView *)tableView {
    return 1;
}
```

Метод `numberOfRowsInSections` должен возвращать количество строк в каждом разделе таблицы. Поскольку стандартная таблица имеет только один раздел, то возвращаемое значение должно соответствовать общему количеству строк в таблице. Свяжите это значение с вашими действительными данными:

```
- (NSInteger)tableView:(UITableView *)tableView numberOf
RowsInSection:(NSInteger)section
{
    return nRecords;
}
```

Наконец, метод `cellForRowAtIndexPath` возвращает объект `UITableViewCell`, содержащий отображаемую информацию для заданной ячейки таблицы. Класс `UITableViewCell` является универсальным классом, поддерживающим текст и изображения, а также включающим функциональность подтверждений на редактирование и удаление.

Каждая ячейка при своем создании кэшируется в памяти, поэтому если она была создана ранее, то вы сможете найти ее в очереди таблицы. Это позволяет прокручивать таблицу, не пересоздавая ячейки всякий раз, когда они попадают в зону видимости. Кроме того, это позволяет таблице избавляться от неиспользуемых ячеек при нехватке памяти, при необходимости пересоздавая их позднее. Для этих целей ваша привязка данных должна быть готова к обслуживанию одних и тех же ячеек множество раз:

```
- (UITableViewCell *)tableView:(UITableView *)tableView
    cellForRowAtIndexPath:(NSIndexPath *)indexPath
```

```
{
    NSString *CellIdentifier = [ [ NSString alloc ] initWithFormat:
        @"Cell %d", [ indexPath indexAtPosition: 1 ] ];

    /* Ищем ячейки в очереди таблицы */
    UITableViewCell *cell =
        [ tableView dequeueReusableCellWithIdentifier: CellIdentifier ];

    /* Не нашли в очереди, создаем объект новой ячейки */
    if (cell == nil) {
        cell = [ [ [ UITableViewCell alloc ]
            initWithFrame: CGRectZero reuseIdentifier: CellIdentifier ]
        autorelease ];
    }

    /* Задаем текст для ячейки */
    cell.text = CellIdentifier;
    return cell;
}
```

Используемый в этом примере индексный путь (index path) ссылается на основной объект `NSIndexPath`. Данный объект разрешает внутри одного объекта задавать не один, а несколько индексов. Обращаясь внутри таблицы к индексу, объект `NSIndexPath` содержит номер раздела и номер ячейки. С помощью метода `indexAtPosition` вы можете запросить каждую позицию индекса. Первая позиция (position 0) всегда указывает на *номер раздела* (section number) ячейки, а вторая позиция (position 1) всегда указывает на *номер строки* (row number) ячейки в рамках заданного раздела. В простейших таблицах имеется только один раздел, поэтому номер раздела всегда будет `0`.

Далее в этой главе мы рассмотрим эти методы более подробно.

Ячейки таблицы

Таблица ссылается на каждую запись как на объект ячейки таблицы. Воспринимайте ячейку таблицы не как просто текст, а как небольшой холст. Класс `UITableViewCell` предоставляет функциональность для настройки ячейки таблицы в соответствии с вашими вкусами и предпочтениями. Ячейки

могут содержать изображения, текст, метки и использовать большое многообразие стилей. Как вы уже видели, ячейки таблицы помещаются в очередь самой таблицей, поэтому вам придется создавать ячейки только при первом их использовании или же в случае, если они были выкинуты из памяти.

Каждой ячейке при создании ставится в соответствие *идентификатор повторного использования* (reuse identifier). Он служит для идентификации ячейки внутри очереди таблицы. В предыдущем примере вы включили в идентификатор номер ячейки, но вы можете задать любое понравившееся вам уникальное значение:

```
NSString *CellIdentifier = [[NSString alloc] initWithString: @"Frank"];
UITableViewCell *cell = [ [ [ UITableViewCell alloc ]
            initWithFrame: CGRectZero
            reuseIdentifier: CellIdentifier
  ] autorelease
];
```

После создания ячейки таблицы вы можете определить для нее целый ряд различных стилевых параметров.

Отображаемый текст

Чтобы добавить в ячейку отображаемый текст, воспользуйтесь свойством `text` ячейки:

```
cell.text = @"Frank's Table Cell";
```

Выравнивание

Настройте выравнивание текста в ячейке путем определения значения свойства `textAlignment`. Вы использовали аналогичные свойства при работе с текстовыми представлениями:

```
cell.textAlignment = UITextAlignmentLeft;
```

По умолчанию задается выравнивание текста по левому краю, но вы можете использовать любое из приведенных далее значений. Это те же самые значения, что и используемые с классом `UITextView`, которые были рассмотрены ранее:

- `UITextAligmentLeft` — текст выравнивается по левому краю (задан по умолчанию);

- `UITextAligmentRight` — текст выравнивается по правому краю;
- `UITextAligmentCenter` — текст выравнивается по центру.

Шрифт и размер

Чтобы определить для ячейки шрифт текста и кегль, присвойте объект `UIFont` свойству `font` текстового представления. Это происходит точно так же, как и задание шрифта для объекта `UITextView`. Чтобы создать объект `UIFont`, импортируйте заголовочный UI Kit-файл UIFont.h:

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

Чтобы без особого труда породить новые шрифты, можете воспользоваться статическим методом `fontWithName`:

```
UIFont *myFont = [ UIFont fontWithName: @"Arial" size: 18.0 ];
cell.font = myFont;
```

Кроме того, для создания системных шрифтов существуют три других статических метода:

```
UIFont *mySystemFont = [ UIFont systemFontOfSize: 12.0 ];
UIFont *myBoldSystemFont = [ UIFont boldSystemFontOfSize: 12.0 ];
UIFont *myItalicSystemFont = [ UIFont italicSystemFontOfSize: 12.0 ];
```

Выбор гарнитуры определяет отображаемый шрифт для всего текста только внутри ячейки. Ячейка таблицы напрямую не поддерживает формат обогащенного текста (rich text).

Цвет текста

Задать цвет текста ячейки можно, назначив объект `UIColor` свойству `textColor` ячейки. Чтобы создать объект `UIColor`, импортируйте заголовочный UI Kit-файл UIColor.h:

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

Для создания объектов цвета можно использовать статические методы, которые автоматически удаляются, когда необходимость в них отпадает. Цвета можно создавать как уровни белого, используя оттенки, или как композицию RGB. Ранее в этой главе вы уже познакомились с цветами.

После создания объекта `UIColor` присвойте значение его свойству `textColor` ячейки:

```
cell.textColor = [ UIColor redColor ];
```

Кроме того, вы можете задать цвет текста для выделенной (выбранной) ячейки с помощью свойства `selectedTextColor`:

```
cell.selectedTextColor = [ UIColor blueColor ];
```

Поскольку ячейка напрямую не поддерживает обогащенный текст (rich text), то выбор цвета влияет на весь текст внутри ячейки.

Изображения

Чтобы добавить в ячейку изображение, присвойте объект `UIImage` свойству `image` ячейки:

```
cell.image = [ UIImage imageNamed: @"cell.png" ];
```

Кроме того, вы можете определить изображение, которое будет отображаться при выделении ячейки:

```
cell.selectedImage = [ UIImage imageNamed: @"selected_cell.png" ];
```

Метод `imageNamed` класса `UIImage` извлекает изображение из папки вашего приложения. Все изображения, используемые в вашей таблице, должны иметь примерно одинаковые размеры, чтобы ваша таблица выглядела единообразно. В зависимости от высоты вашего изображения вам может потребоваться настроить высоту строки таблицы. Вы можете установить для всей таблицы стандартную высоту строки, подменив метод `init` контроллера табличного представления, чтобы задать свойство `rowHeight`:

```
- (id)init {
    self = [ super init ];
    if (self != nil) {
        self.tableView.rowHeight = 65;
    }
    return self;
}
```

Если же вы пожелаете иметь разную высоту строк, то можете задать высоту для каждой ячейки по отдельности, подменив метод `heightForRowAtIndexPath` внутри источника данных:

```
- (CGFloat)tableView:(UITableView *)tableView
    heightForRowAtIndexPath:(NSIndexPath *)indexPath
{
    if ([ indexPath indexAtPosition: 1 ] == 0)
```

```
        return 65.0;
    else
        return 40.0;
}
```

Стиль выбора

Путем изменения свойства `selectionStyle` вы можете определить цвет, используемый при выделении ячейки:

```
cell.selectionStyle = UITableViewCellSelectionStyleBlue;
```

По умолчанию ячейка выделяется синим цветом, но могут быть и другие варианты:

- `UITableViewCellSelectionStyleBlue` — выделение выбранных ячеек синим цветом;
- `UITableViewCellSelectionStyleGray` — выделение выбранных ячеек серым цветом;
- `UITableViewCellSelectionStyleNone` — выбранные ячейки не выделяются.

Метки

Метки (labels) являются миниатюрными классами представлений, которые могут быть добавлены к ячейкам таблицы, чтобы добавить ячейке таблицы еще больше декорирующего текста. Метки применяются для добавления в ячейку периферийного текста, такого как дата (day), тема (subject) и эскиз (preview), используемые в приложении Mail от Apple.

Метка инициализируется со смещением области отображения относительно ячейки. Приведенный далее пример создает метку со сдвигом (100, 0) пикселов внутри ячейки таблицы и имеющую размер 50×50 пикселов:

```
UILabel *label = [ [ UILabel alloc ] initWithFrame:
    CGRectMake(100.0, 0.0, 50.0, 50.0)
];
```

Текст для метки задайте с помощью свойства `text`:

```
label.text = @"Label Text";
```

Метка использует многие из свойств текста, что и представление ячейки или текстовое представление, включая выравнивание текста, цвет и шрифт:

```
label.textAlignment = UITextAlignmentLeft;
label.textColor = [ UIColor redColor ];
label.font = [ UIFont fontWithName: @"Arial" size: 10.0 ];
```

Кроме того, метка разрешает вам применять затенение текста. Вы можете даже определить смещение тени, передав структуру `CGSize` свойству `shadowOffset`:

```
label.shadowColor = [ UIColor grayColor ];
label.shadowOffset = CGSizeMake(0, -1);
```

Метки также позволяют определять цвет текста для выделения ячейки:

```
label.highlightedTextColor = [ UIColor blackColor ];
```

Если вы хотите еще больше выделить вашу метку, то можете задать даже цвет фона. Цвет фона будет применен только к области отображения метки. Это может пригодиться для отладки при ручной расстановке меток или для отпугивания ваших заказчиков:

```
label.backgroundColor = [ UIColor blueColor ];
```

После создания метки прикрепите ее к ячейке в качестве подчиненного представления:

```
[ cell addSubview: label ];
```

Раскрытия

Раскрытия (disclosures), называемые также аксессуарами (accessories) — 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, а в конце уже песню.

Помимо стрелок для обозначения тех ячеек, которые были выделены в таблицах, разрешающих множественное выделение, можно отображать флажки.

Любая заданная ячейка может отобразить аксессуар, установив свойство `accessoryType` ячейки:

```
cell.accessoryType = UITableViewCellAccessoryDisclosureIndicator;
```

Доступны следующие стили аксессуаров (табл. 3.8).

Таблица 3.8

Стиль	Описание
UITableViewCellAccessoryNone	Нет аксессуара
UITableViewCellAccessoryDisclosureIndicator	Шеврон черной стрелки
UITableViewCellAccessoryDetailDisclosureButton	Синяя кнопка раскрытия
UITableViewCellAccessoryCheckmark	Флажок для выбора

Реализация множественного выбора

Как вы уже знаете, ячейка таблицы может отобразить флажок посредством свойства accessoryType. Когда пользователь выбирает ту или иную ячейку, вызывается метод didSelectRowAtIndexPath таблицы-делегата. Этот метод является частью протокола UITableViewDelegate. Добавление этого метода к вашему делегату позволит вам добавить к вашей таблице поддержку множественного выбора путем управления, для каких ячеек установлен аксессуар флажка:

```
- (void)tableView:(UITableView *)tableView
    didSelectRowAtIndexPath:(NSIndexPath *)indexPath
{
    NSLog(@"Selected section %d, cell %d",
          [ indexPath indexAtPosition: 0 ],
          [ indexPath indexAtPosition: 1 ]);

    /* Получаем указатель на выделенную ячейку таблицы */
    UITableViewCell *cell =
     [ self.tableView cellForRowAtIndexPath: indexPath ];

    /* Переключаем дополнительный тип */
    if (cell.accessoryType == UITableViewCellAccessoryNone)
        cell.accessoryType = UITableViewCellAccessoryCheckmark;
    else
        cell.accessoryType = UITableViewCellAccessoryNone;
}
```

Редактирование и действие "провести, чтобы удалить"

Чтобы разрешить пользователям удалять объекты в таблице, разрешите функцию редактирования таблицы. Это приведет к тому, что каждая ячейка в таблице будет отображать в левом углу красный значок удаления. Во время редактирования каждая ячейка автоматически получит соответствующий отступ:

```
[ self.tableView setEditing:YES animated:YES ];
```

Это действие должно быть инициировано действием кнопки панели навигации, например, кнопкой **Edit**, или какой-либо другой с аналогичным действием. Чтобы разрешить пользователю после завершения редактирования оставить режим редактирования, воспользуйтесь тем же методом:

```
[ self.tableView setEditing: NO animated: YES ];
```

Во время процесса редактирования пользователь может удалить какую-либо запись из таблицы. В этом случае пользователю будет отображено подтверждение на удаление. После получения подтверждения уведомляется метод `commitEditingStyle` источника данных, чтобы проинформировать ваше приложение, что был получен запрос на удаление. Этот запрос так же принадлежит протоколу `UITableViewDataSource`. Вы сами решаете, как обработать запрос и удалить соответствующие данные из вашего источника данных. Также вы проинструктируете табличное представление, как удалить строку:

```
- (void)tableView:(UITableView *)tableView
    commitEditingStyle:(UITableViewCellEditingStyle) editingStyle
    forRowAtIndexPath:(NSIndexPath *) indexPath
{
    if (editingStyle == UITableViewCellEditingStyleDelete) {
        NSLog(@"Deleted section %d, cell %d",
              [ indexPath indexAtPosition: 0 ],
              [ indexPath indexAtPosition: 1 ]);

        /* Дополнительный код для удаления ячейки из ваших данных */

        /* Удаляем ячейку из таблицы */

        NSMutableArray *array = [ [ NSMutableArray alloc ] init ];
        [ array addObject: indexPath ];
```

```
        [ self.tableView deleteRowsAtIndexPaths: array
            withRowAnimation: UITableViewRowAnimationFade
        ];
    }
}
```

Примечание

Если в вашем классе-делегате существует такой метод, то функциональность "провести, чтобы удалить" станет активной автоматически.

Метод `deleteRowsAtIndexPath` позволяет вам удалять из таблицы одну или более строк, передавая массив индексов. Вы также можете задать одну из нескольких предопределенных анимаций, которую вы бы хотели использовать для удаления ячеек. Поддерживаются следующие анимации (табл. 3.9).

Таблица 3.9

Анимация	Описание
`UITableViewRowAnimationFade`	Ячейки исчезают
`UITableViewRowAnimationRight`	Ячейки выскальзывают справа
`UITableViewRowAnimationLeft`	Ячейки выскальзывают слева
`UITableViewRowAnimationTop`	Ячейки выскальзывают сверху смежной ячейки
`UITableViewRowAnimationBottom`	Ячейки выскальзывают снизу смежной ячейки

Перезагрузка таблицы

Если данные внутри вашей таблицы изменились, то перезагрузите таблицу, вызвав метод `reloadData` табличного представления. Снова будет запрошен источник данных, и любые внесенные вами в ваши данные изменения будут переданы в таблицу:

```
[ self.tableView reloadData ];
```

Это лучше делать исключительно тогда, когда данные таблицы были изменены с момента последнего взаимодействия с ней пользователя, и не рекомен-

дуется перезагружать таблицу после удаления ячейки. При перезагрузке таблицы разрушается и пересоздается заново структура всей таблицы, поэтому вам лучше быть осторожными при использовании этого метода. Для удаления отдельных ячеек воспользуйтесь методом `deleteRowsAtIndexPath` для анимации процесса удаления одной или нескольких ячеек.

Простейший проводник файлов: TableDemo

Пример TableDemo отображает таблицу, содержащую список файлов и каталогов, имеющихся в домашней папке вашего приложения на iPhone (рис. 3.4). Для предоставления пользователю кнопок редактирования и кнопки перезагрузки данный пример использует контроллер табличного представления, прикрепленный к контроллеру панели навигации. Пользователь сможет нажать кнопку **Delete** для удаления элементов из списка (не волнуйтесь, на самом деле никакие файлы удалены не будут). Функциональность "провести, чтобы удалить" также будет активна, позволяя пользователю проводить по ячейкам, чтобы удалить их.

Рис. 3.4. Пример TableDemo

Вы можете скомпилировать это приложение, приведенное в листингах 3.32—3.36, с помощью SDK, создав проект приложения TableDemo на базе представления. Если вы хотите увидеть, как создавать все эти объекты с нуля, то удалите код Interface Builder.

Листинг 3.32. Прототипы делегата приложения TableDemo (TableDemoAppDelegate.h)

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

@class TableDemoViewController;

@interface TableDemoAppDelegate : NSObject <UIApplicationDelegate> {
    UIWindow *window;
    TableDemoViewController *viewController;
    UINavigationController *navigationController;
}

@property (nonatomic, retain) IBOutlet UIWindow *window;
@property (nonatomic, retain) IBOutlet TableDemoViewController
  *viewController;

@end
```

Листинг 3.33. Делегат приложения TableDemo (TableDemoAppDelegate.m)

```
#import "TableDemoAppDelegate.h"
#import "TableDemoViewController.h"

@implementation TableDemoAppDelegate

@synthesize window;
@synthesize viewController;

- (void)applicationDidFinishLaunching:(UIApplication *)application {
    CGRect screenBounds = [ [ UIScreen mainScreen ] bounds ];
```

```
    self.window = [ [ [ UIWindow alloc ] initWithFrame: screenBounds ]
      autorelease ];
    viewController = [ [ TableDemoViewController alloc ] init ];
    navigationController = [ [ UINavigationController alloc ]
      initWithRootViewController: viewController ];

    [ window addSubview: [ navigationController view ] ];
    [ window makeKeyAndVisible ];
}

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

@end
```

Листинг 3.34. Прототип контроллера представлений TableDemo (TableDemoViewController.h)

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

@interface TableDemoViewController : UITableViewController {
    NSMutableArray *fileList;
}

- (void) startEditing;
- (void) stopEditing;
- (void) reload;

@end
```

Листинг 3.35. Контроллер представлений TableDemo (TableDemoViewController.m)

```
#import "TableDemoViewController.h"

@implementation TableDemoViewController
```

```
- (id)init {
    self = [ super init ];

    if (self != nil) {
        /* Создаем список файлов */
        [ self reload ];

        /* Инициализируем навигационные кнопки */
        self.navigationItem.rightBarButtonItem
        = [ [ [ UIBarButtonItem alloc ]
             initWithBarButtonSystemItem: UIBarButtonSystemItemEdit
             target: self
             action: @selector(startEditing) ] autorelease ];

        self.navigationItem.leftBarButtonItem
        = [ [ [ UIBarButtonItem alloc ]
             initWithTitle:@"Reload"
             style: UIBarButtonItemStylePlain
             target: self
             action:@selector(reload) ]
           autorelease ];
    }
    return self;
}

- (void) startEditing {
    [ self.tableView setEditing: YES animated: YES ];

    self.navigationItem.rightBarButtonItem =
      [ [ [ UIBarButtonItem alloc ]
       initWithBarButtonSystemItem: UIBarButtonSystemItemDone
       target: self
       action: @selector(stopEditing) ] autorelease ];
}

- (void) stopEditing {
    [ self.tableView setEditing: NO animated: YES ];
```

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

- (void) reload {
    NSDirectoryEnumerator *dirEnum;
    NSString *file;

    fileList = [ [ NSMutableArray alloc ] init ];
    dirEnum = [ [ NSFileManager defaultManager ] enumeratorAtPath:
        NSHomeDirectory()
    ];

    while ((file = [ dirEnum nextObject ])) {
        [ fileList addObject: file ];
    }

    [ self.tableView reloadData ];
}

- (NSInteger)numberOfSectionsInTableView:(UITableView *)tableView {
    return 1;
}

- (NSInteger)tableView:(UITableView *)tableView
     numberOfRowsInSection:(NSInteger)section {
    return [ fileList count ];
}

- (UITableViewCell *)tableView:(UITableView *)tableView
     cellForRowAtIndexPath:(NSIndexPath *)indexPath {

    NSString *CellIdentifier = [ fileList objectAtIndex:
        [ indexPath indexAtPosition: 1 ]
    ];
```

```
    UITableViewCell *cell = [ tableView
        dequeueReusableCellWithIdentifier: CellIdentifier
    ];

    if (cell == nil) {
        cell = [ [ [ UITableViewCell alloc ] initWithFrame:
            CGRectZero reuseIdentifier: CellIdentifier ] autorelease
        ];
        cell.text = CellIdentifier;

        UIFont *font = [ UIFont fontWithName: @"Courier" size: 12.0 ];
        cell.font = font;
    }
    return cell;
}

- (void)tableView:(UITableView *)tableView
    commitEditingStyle:(UITableViewCellEditingStyle) editingStyle
    forRowAtIndexPath:(NSIndexPath *) indexPath
{
    if (editingStyle == UITableViewCellEditingStyleDelete) {
      /* Удаляем ячейку из источника данных */

      UITableViewCell *cell = [ self.tableView cellForRowAtIndexPath:
            indexPath ];

      for (int i = 0; i < [ fileList count ]; i++) {
        if ([cell.text isEqualToString: [fileList objectAtIndex: i]]) {
            [ fileList removeObjectAtIndex: i ];
        }
      }

      /* Удаляем ячейку из таблицы */
      NSMutableArray *array = [ [ NSMutableArray alloc ] init ];
      [ array addObject: indexPath ];
```

```
      [ self.tableView deleteRowsAtIndexPaths: array
            withRowAnimation: UITableViewRowAnimationTop
      ];
    }
}

- (void)tableView:(UITableView *)tableView didSelectRowAt
    IndexPath:(NSIndexPath *)indexPath {

    UITableViewCell *cell = [ self.tableView cellForRowAtIndexPath:
      indexPath ];

    UIAlertView *alert = [ [ UIAlertView alloc ] initWithTitle:
      @"File Selected"
        message: [ NSString stringWithFormat:
          @"You selected the file '%@'",
        cell.text ]
        delegate: nil
        cancelButtonTitle: nil
        otherButtonTitles: @"OK", nil
    ];

    [ alert show ];
}

- (void)loadView {
    [ super loadView ];
}

- (BOOL)shouldAutorotateToInterfaceOrientation:
    (UIInterfaceOrientation)interfaceOrientation {
    return YES;
}

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

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

@end
```

Листинг 3.36. Функция `main` для TableDemo (main.m)

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

int main(int argc, char *argv[]) {

NSAutoreleasePool * pool = [[NSAutoreleasePool alloc] init];
    int retVal = UIApplicationMain(argc, argv, nil,
                                   @"TableDemoAppDelegate");
    [pool release];
    return retVal;
}
```

Как это работает

Пример TableDemo работает следующим образом:

1. Приложение порождается точно так же, как и все предыдущие примеры, и вызывается программная функция `main`, которая вызывает метод `applicationDidFinishLaunching` класса `TableDemoAppDelegate` и создает соответствующее окно, контроллер представлений и классы контроллера навигации.
2. Контроллер представлений создается как экземпляр `UITableViewController`. Метод `init` контроллера подменяется для загрузки в массив `fileList` списка файлов. Также он добавляет кнопки панели навигации: системную кнопку **Edit** и кнопку **Reload**.
3. Когда таблица визуализирована, автоматически запрашиваются методы источника данных контроллера. Метод `numberOfRowsInSection` возвращает количество строк в списке файлов. Метод `cellForRowAtIndexPath` создает новую ячейку, используя в качестве заголовка ячейки имя файла.

4. Если пользователь касается кнопки **Edit**, то уведомляется назначенный кнопке селектор, `startEditing`. Это приводит к замене кнопки на кнопку **Done** и разрешению редактирования таблицы. Таблица автоматически создает соответствующие отступы для ячеек и добавляет к ним значки удаления.
5. Если пользователь удалит какую-либо строку либо путем редактирования, либо методом "провести, чтобы удалить", у него будет запрошено подтверждение. После получения подтверждения запрос на удаление приведет к уведомлению о запросе метода `commitEditingStyle`. Этот метод проверит, является ли полученный запрос запросом на удаление, и удалит объект из списка файлов и таблицы.
6. Если пользователь нажмет кнопку **Reload**, то список файлов будет повторно прочтен, а таблица будет обновлена.

Для дальнейшего изучения

Посмотрим, можете ли вы взять что-либо из ранее изученного и применить к данному примеру.

- Используйте метод `fileAttributes` класса `NSDirectoryEnumerator` для определения того, какие файлы являются каталогами. К ячейкам, обозначающим каталоги, добавьте стрелку раскрытия и, когда пользователь коснется одной из них, протолкните контроллер нового табличного представления в стек. Контроллер дочерних представлений должен иметь кнопку **Back** и отображать только те файлы, которые присутствуют в выбранной папке.
- Добавьте новый контроллер представлений, отображающий содержимое какого-либо файла в `UITextView`. Если этот файл является бинарным, то отобразите вывод в шестнадцатеричном формате. Когда пользователь коснется файла, то необходимо протолкнуть новый контроллер представлений в стек контроллера навигации, отображающего содержимое файла. Новый контроллер представлений должен иметь кнопку **Back**.
- В заголовочных файлах вашего SDK проверьте наличие следующих прототипов: UITableView.h, UITableViewCell.h, UILabel.h и UITableViewController.h. Вы найдете их в папке /Developer/Platforms/iPhoneOS.platform внутри каталога Headers библиотеки UI Kit.

Манипуляции строкой состояния

Внешний вид строки состояния можно настраивать, чтобы она соответствовала духу и назначению вашего приложения, или же ее можно вовсе удалить, если возникнет необходимость в дополнительном пространстве экрана. Вносимые в строку состояния изменения затрагивают только ваше приложение, стандартная строка состояния вернет все в исходное состояние, когда вы нажмете кнопку **Home**.

Вы можете изменить многие свойства строки состояния с помощью методов класса `UIApplication`. Класс `UIApplication` представляет ваше запущенное приложение. Чтобы получить доступ к этому объекту, воспользуйтесь статическим методом `sharedApplication` этого класса:

```
UIApplication *myApp = [ UIApplication sharedApplication ];
```

Скрытие строки состояния

Чтобы полностью скрыть строку состояния, воспользуйтесь методом `setStatusBarHidden`:

```
[ myApp setStatusBarHidden: YES animated: YES];
```

Если вы собираетесь скрыть строку состояния, то убедитесь в том, что сделали это до того, как создали окна или представления. В противном случае рамка приложения не будет включать пространство, занятое строкой состояния, а вам придется восстанавливать все задействованные классы представлений с помощью метода `setFrame`:

```
[ viewController.view setFrame: [ [ UIScreen mainScreen ]
   applicationFrame ] ];


[ [ navigationController view ]
    setFrame: [ [ UIScreen mainScreen ] applicationFrame ]
];
```

Стиль строки состояния

Стиль строки состояния определяет ее цвет и прозрачность. Стиль строки состояния задает весь ее внешний вид:

```
[ myApp setStatusBarStyle: UIStatusBarStyleBlackOpaque ];
```

Поддерживаются следующие стили строки состояния (табл. 3.10).

Таблица 3.10

Стиль	Описание
`UIStatusBarStyleDefault`	По умолчанию; строка состояния белого цвета, текст черного цвета
`UIStatusBarStyleBlackTranslucent`	Черная прозрачная строка состояния, текст белого цвета
`UIStatusBarStyleBlackOpaque`	Черная непрозрачная строка состояния, текст белого цвета

Ориентация строки состояния

Обычно ориентация строки состояния меняется автоматически при использовании контроллеров представлений. Чтобы самостоятельно изменить ориентацию строки состояния, воспользуйтесь методом `setStatusBarOrientation` класса `UIApplication`:

```
[ myApp setStatusBarOrientation: UIInterfaceOrientationLandscapeRight
    animated: NO
];
```

Для iPhone существуют следующие ориентации:

- `UIDeviceOrientationPortrait` — устройство ориентировано прямо по вертикали в книжном режиме;
- `UIDeviceOrientationPortraitUpsideDown` — устройство ориентировано вверх дном по вертикали в книжном режиме;
- `UIDeviceOrientationLandscapeLeft` — устройство повернуто против часовой стрелки в альбомном режиме;
- `UIDeviceOrientationLandscapeRight` — устройство повернуто по часовой стрелке в альбомном режиме;
- `UIDeviceOrientationFaceUp` — устройство лежит на ровной поверхности лицевой стороной вверх, как, например, на столе;
- `UIDeviceOrientationFaceDown` — устройство лежит на ровной поверхности лицевой стороной вниз, как, например, на столе.

Бейджи приложения

Имея все эти многочисленные подключения — 3G/EDGE, Wi-Fi и Bluetooth, 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. После того как во встроенных в iPhone программах появились новые функции, такие как выскакивающее уведомление (push notification), у вашего приложения появилась возможность получать новую информацию, если оно не находится в фоновом режиме. Помимо этого, бейджи приложения могут служить напоминаниями пользователю, который запустил какое-либо приложение, но еще не закончил просмотр всех сообщений, уведомлений или других данных. Бейджи приложений (application badges) — это небольшие кружки сообщений, отображаемые на значке домашнего экрана программы. Бейджи приложений активно используются поставляемыми Apple приложениями для уведомления пользователя о пропущенных звонках, полученных голосовых или текстовых сообщениях и 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, 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 sharedApplication ].applicationIconBadgeNumber = 42;
```

Свойство `applicationIconBadgeNumber` принимает объект `NSInteger`, поэтому вы можете задавать с его помощью только целочисленные значения, но этого зачастую вполне достаточно для передачи сообщения о том, что определенное количество элементов ожидает пользовательского внимания.

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. Бейдж удаляется, когда его значение устанавливается в ноль. Такой код лучше всего поместить после реализации перехода к управляющему представлению, отображающему новые события:

```
[ UIApplication sharedApplication ].applicationIconBadgeNumber = 0;
```

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, установите это свойство в методе `applicationWillTerminate` делегата приложения:

```
- (void)applicationWillTerminate {
    /* Мы собираемся выходить, поэтому удалите бейдж приложения */

    [ UIApplication sharedApplication ].applicationIconBadgeNumber = 0;
}
```

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 цифр, которое может быть добавлено на бейдж приложения. Что происходит, когда вы превышаете этот лимит?
- В заголовочных файлах вашего SDK проверьте наличие прототипа UIApplication.h. Вы найдете его в папке /Developer/Platforms/iPhoneOS.platform внутри каталога Headers библиотеки UI Kit.

Сервисы приложения

Состояние приложения имеет большее значение на iPhone, нежели на настольном компьютере. Это связано с тем, что многие события на iPhone мо-

гут приводить к приостановке работы приложения или к 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. Хотя обычно приложение ничего не может сделать относительно того состояния, в которое оно переходит, но оно может, по крайней мере, предпринять какие-либо действия для подготовки к этому.

Сервисы приложения (application services) предоставляются путем уведомления класса-делегата приложения о событиях, которые вот-вот произойдут. Все методы, перечисленные в этом разделе, принадлежат протоколу `UIApplicationDelegate`. Делегат приложения задается при первом порождении части приложения на Objective-C путем вызова `UIApplicationMain` в вашем файле main.m. Если вы используете nib-файл, то описываете делегат приложения внутри шаблона Interface Builder.

Приостановка и возобновление

При блокировке устройства или поступлении входящего звонка приложения SDK приостанавливают свою работу. Когда это происходит, уведомляется делегат метода `applicationWillResignActive`. Чтобы подготовить приложение к тому, что оно может быть приостановлено, например, осуществить закрытие всех сетевых соединений или сохранение данных, можно подменить этот метод:

```
- (void)applicationWillResignActive:(UIApplication *) application {
    NSLog(@"About to be suspended");

    /* Код для подготовки к приостановке */
}
```

Пока ваше приложение приостановлено, оно *не будет выполняться в фоновом режиме*. Когда приложение возобновляет свою работу, уведомляется другой метод — `applicationDidBecomeActive`. Здесь вы можете добавить код, чтобы продолжить работу с того места, на котором ваше приложение остановилось:

```
- (void)applicationDidBecomeActive:(UIApplication *) application {
    NSLog(@"Became active");

    /* Код для подготовки к возобновлению */
}
```

ВНИМАНИЕ!

Когда приложение начинает свою работу, после его метода `applicationDidFinishLaunching` уведомляется еще и его метод `applicationDidBecomeActive`. Убедитесь в том, что ваш код может уловить разницу между возобновлением (resuming) и началом (starting) работы.

Прекращение работы программы

Программа прекращает (terminate) свою работу, когда пользователь нажимает кнопку **Home** или выключает iPhone. Метод `applicationWillTerminate` вызывается всякий раз, когда приложение собирается аккуратно завершить свою работу. Он *не* вызывается, если приложение убивается путем удерживания кнопки **Home**. 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:(UIApplication *)application {
    /* Код очистки ресурсов и завершения работы */
}
```

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`. Ранее в этой главе вы уже познакомились с `NSURL`. Объект `NSURL` передается в метод `openURL` приложения, где окружение приложения обрабатывает и запускает соответствующее приложение-обработчик:

```
NSURL *url = [ [ NSURL alloc ] initWithString:
    @"http://www.oreilly.com/" ];
[ [ UIApplication sharedApplication ] openURL: url ];
```

Инициация телефонных звонков

Как было только что продемонстрировано, метод `openURL` вызывает Safari для запуска URL Web-узлов. На самом же деле происходит следующее: каждый протокол сопоставлен отдельному приложению-обработчику. 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 служит протокольный префикс `tel:`:

```
NSURL *url = [ [ NSURL alloc ]
    initWithString: @"tel:212-555-1234" ];
[ [ UIApplication sharedApplication ] 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.

Глава 4

События множественных касаний и геометрия

В *главе 3* были рассмотрены основные элементы пользовательского интерфейса iPhone. Многие из объектов поддерживают события высокого уровня, уведомляющие приложение об определенных 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 обработки основных событий мыши, которые происходят, когда пользователь касается или проводит по экрану iPhone. Высокоуровневые объекты, например, таблицы и листы действий, берут эти события низкого уровня и оборачивают их в высокоуровневые, чтобы обрабатывать нажатия кнопок мыши, выбор строки и другие действия. Apple предоставляет API множественных касаний (multi-touch), которое способно перехватывать жесты пальцев, чтобы воспроизвести аналогичное применение перемещений пальцев в вашем пользовательском интерфейсе. API *касаний* (touches) абсолютно четко сообщает приложению, что произошло на экране, и предоставляет информацию, необходимую приложению для взаимодействия с пользователем.

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*. Библиотека Core Graphics предоставляет множество основных структур для управления функциями, отвечающими 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.

Структура *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 структуры `CGPoint` используется метод `CGPointMake`:

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

Первое значение соответствует *X*-координате по горизонтали, а 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 — точка (319, 479); отсчет координат идет от нуля.

Будучи универсальной структурой, `CGPoint` может одинаково хорошо ссылаться как на координату на экране, так и на координату в области окна. Например, если окно имеет размеры 320×240 (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);
```

Структура *CGSize*

Структура `CGSize` представляет размер прямоугольника. Она инкапсулирует ширину и высоту объекта и изначально находится в API iPhone, задавая раз-

мер объектов экрана, т. е. окон. Чтобы создать объект `CGSize`, воспользуйтесь методом `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);
```

Структура *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(0.0, 200.0, 320.0, 240.0);
```

Этот пример описывает окно 320×240, левый верхний угол которого расположен в точке (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 применяться для определения, содержит ли одна структура `CGRect` другую структуру `CGRect`. Это полезно при выяснении, перекрываются ли заданные объекты:

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

Чтобы одна структура содержала другую, необходимо, чтобы все пикселы одной структуры были и в другой. А чтобы две структуры пересекались, необходимо чтобы у них был общим хотя бы один пиксел. 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.

Обработка событий множественных касаний

Поддержка множественных касаний в iPhone предоставляет набор событий касаний, состоящих из небольших отдельных частей единичного множественного жеста. Например, если вы поместили свой палец на экране, то ге-

нерируется одно событие, если вы поместите второй палец, будет сгенерировано другое событие, а если вы переместите оба пальца, то будет сгенерировано еще одно событие. Все эти события обрабатываются посредством API `UITouch` и `UIEvent`. Эти объекты предоставляют информацию о том, какие жесты были сделаны, и координаты места, где это произошло. Место касания находится в соответствии с конкретным окном или представлением, поэтому вы также узнаете о том, как сообщить, в каком объекте произошло касание.

Поскольку события относятся к объекту, в котором они произошли, то возвращаемые координаты будут на самом деле не абсолютными координатами экрана, а относительными координатами. Например, если окно представления отображено на половине экрана с началом в точке (0, 240), а ваш пользователь коснулся левого верхнего угла этого представления, то переданные в ваше приложение координаты будут (0, 0), а не (0, 240). Координаты (0, 0) соответствуют левому верхнему углу объекта представления, которого коснулся пользователь.

Уведомления *UITouch*

Класс `UITouch` является классом, используемым для передачи одного действия из набора жестов. Объект `UITouch` может включать уведомления об одном или более опущенном пальце, перемещающемся пальце или прекращении жеста. За период единичного жеста может быть получено несколько уведомлений `UITouch`.

Объект `UITouch` содержит различные свойства, идентифицирующие событие:

- `timestamp` предоставляет временную метку относительно предыдущего события с помощью основного класса `NSTimeInterval`;
- `phase` — тип произошедшего события касания. Тем самым приложение информируется о том, что пользователь на самом деле сделал с помощью своего пальца, и может быть одним из следующих значений:
  - `UITouchPhaseBegan` отправляется, когда один или более пальцев коснулись поверхности экрана;
  - `UITouchPhaseMoved` отправляется, когда палец перемещается по экрану. Если пользователь делает жест или перетаскивает какой-либо элемент, то этот тип уведомления посылается несколько раз, обновляя координаты перемещения;

- `UITouchPhaseEnded` отправляется, когда палец убирается с экрана;
- `UITouchPhaseStationary` отправляется, когда палец касается экрана, но не перемещается с момента предыдущего уведомления;
- `UIPhaseCancelled` отправляется, когда система отменяет жест. Это может произойти во время событий, которые могут привести к приостановке работы вашего приложения, например, входящий звонок, или же когда контакта пальца с поверхностью экрана недостаточно, чтобы iPhone посчитал это жестом, и поэтому отменил. Apple в шутку приводит пример прилепления iPhone на лицо, которое приведет к генерации такого уведомления, что может быть полезно, если вы пишете программу "приклей IPhone на лицо" в дополнение к вашему приложению "фонарик";

- `tapCount` определяет количество касаний (включая текущее касание). Например, для второго касания двойного касания это свойство будет иметь значение `2`;
- `window` — указатель на объект `UIWindow`, в котором это событие касания произошло;
- `view` — указатель на объект `UIView`, в котором это событие касания произошло.

Помимо этих свойств класс `UITouch` содержит приведенные далее методы, которые вы можете использовать для определения координат экрана точки, в которой данное событие имело место. Помните, что это относительные координаты для объектов представлений, в которых произошли события, и они не являются действительными координатами экрана:

- `locationInView`

  ```
  - (CGPoint) locationInView: (UIView *) view;
  ```

  Возвращает структуру `CGPoint`, содержащую координаты (относительно объекта `UIView`), в которых произошло данное событие.

- `previousLocationInView`

  ```
  - (CGPoint) previousLocationInView: (UIView *) view;
  ```

  Возвращает структуру `CGPoint`, содержащую координаты (относительно объекта `UIView`), в которых данное событие начиналось. Например, если событие описывает событие `UITouchPhaseMoved`, то оно вернет координаты той точки, с которой палец начал свое перемещение.

Объект *UIEvent*

Объект `UIEvent` объединяет множество объектов `UITouch` в единый переносимый объект, которым вы легко можете управлять. `UIEvent` предоставляет методы для отслеживания касаний, произошедших в одном окне, представлении или даже во всем приложении.

Объекты событий отправляются, как только совершается тот или иной жест, и содержат все события для данного жеста. Основной класс `NSSet` используется для доставки коллекции последующих событий `UITouch`. Каждое из событий `UITouch`, включенное в данное множество, содержит специальную временную метку, фазу и местоположение каждого события, в то время как объект `UIEvent` также содержит собственное свойство временной метки.

Объект `UIEvent` поддерживает следующие методы:

- `allTouches`

  ```
  - (NSSet *) allTouches;
  ```

  Возвращает множество всех касаний, произошедших внутри приложения.

- `touchesForWindow`

  ```
  - (NSSet *) touchesForWindow: (UIWindow *) window;
  ```

  Возвращает множество всех касаний, произошедших только внутри заданного объекта `UIWindow`.

- `touchesForView`

  ```
  - (NSSet *) touchesForView: (UIView *) view;
  ```

  Возвращает множество всех касаний, произошедших только внутри заданного объекта `UIView`.

Обработка событий

Когда пользователь делает какой-либо жест, приложение уведомляет ключевое окно и передает структуру `UIEvent`, содержащую происходящие события. Ключевое окно передает эту информацию первому респонденту для данного окна. Обычно это то представление, в котором произошло реальное событие. После того как жест был сопоставлен с заданным представлением, все последующие события, относящиеся к данному жесту, будут передаваться данному представлению. Объекты `UIApplication` и `UIWindow` содержат метод `sendEvent`:

```
- (void) sendEvent: (UIEvent *)event;
```

Этот метод вызывается как часть процесса диспетчеризации событий и отвечает за получение и отправку событий по соответствующим им адресатам. Вообще говоря, нет необходимости подменять этот метод, если только вы не хотите отслеживать все входящие события.

Когда ключевое окно получает какое-либо событие, оно опрашивает каждый из управляемых им классов представлений, чтобы определить тот, в котором данное событие было порождено, а затем перенаправляет ему это событие. Далее респондент объекта направляет данное событие контроллеру представлений, если представление сопоставлено ему, а затем уже собственному суперклассу представления. Затем оно совершает свой обратный путь по цепочке респондента к своему окну и, в конце концов, к приложению.

Чтобы получать события множественных касаний, вам необходимо в вашем объекте, производном от `UIView`, подменить не менее одного из приведенных далее методов обработчика событий. Ваши классы `UIWindow` и `UIApplication`, чтобы получать события, также могут подменить эти методы, но необходимость в этом возникает крайне редко. Apple для получения событий множественных касаний описала приводимые далее прототипы. Каждый из используемых методов сопоставлен одной из фаз касания, описанных в предыдущем разделе:

```
- (void)touchesBegan:(NSSet *)touches withEvent:(UIEvent *)event;
- (void)touchesMoved:(NSSet *)touches withEvent:(UIEvent *)event;
- (void)touchesEnded:(NSSet *)touches withEvent:(UIEvent *)event;
- (void)touchesCancelled:(NSSet *)touches withEvent:(UIEvent *)event;
```

Каждый метод принимает по два аргумента. Аргумент `NSSet` содержит множество, состоящее только из новых касаний, произошедших с момента последнего события. Аргумент `UIEvent` имеет две цели: он позволяет вам отфильтровывать события касаний, относящиеся исключительно к конкретному представлению или окну, а также он предоставляет средства организации доступа к предыдущим касаниям, если они принадлежат текущему жесту.

Вам необходимо будет реализовать только методы уведомлений для интересующих вас событий. Если вам нужно получать только касания, похожие на "щелчки мыши", то следует использовать только методы `touchesBegan` и `touchesEnded`. Если вы пользуетесь перетаскиванием, как, например, в элементе управления бегунком, то вам потребуется метод `touchesMoved`. Метод `touchesCancelled` является необязательным, но Apple рекомендует использовать его для очистки объектов в постоянных классах.

Пример: счетчик касаний

В этом примере вы подмените метод `touchesBegan` для определения единичного, двойного и тройного касаний. Для наших демонстрационных целей нам будет достаточно направить вывод только на консоль, но в вашем собственном приложении вы направите эту информацию к компонентам пользовательского интерфейса.

Чтобы скомпоновать это приложение, создайте в Xcode приложение на базе представления (view-based application) TouchDemo и добавьте приводимый далее код в ваш класс TouchDemoViewController.m:

```
- (void) touchesBegan:(NSSet *)touches withEvent:(UIEvent *)event
{
    UITouch *touch = [ touches anyObject ];
    CGPoint location = [ touch locationInView: self.view ];
    NSUInteger taps = [ touch tapCount ];

    NSLog(@"%s tap at %f, %f tap count: %d",
        (taps == 1) ? "Single" :
        (taps == 2) ? "Double" : "Triple+",
        location.x, location.y, taps);
}
```

Скомпилируйте это приложение и запустите в симуляторе. Чтобы отобразить консоль, в Xcode в меню **Run** выберите команду **Console**. Теперь поэкспериментируйте с единичным, двойным и тройным касаниями на экране симулятора. В консоли вы увидите вывод, схожий с приведенным далее:

```
Untitled[4229:20b] Single tap at 161.000000, 113.953613 taps count: 1
Untitled[4229:20b] Double tap at 161.000000, 113.953613 taps count: 2
Untitled[4229:20b] Triple+ tap at 161.000000, 113.953613 taps count: 3
Untitled[4229:20b] Triple+ tap at 161.000000, 113.953613 taps count: 4
```

Примечание

Подсчет касаний будет увеличиваться бесконечно, т. е. чисто технически вы можете отслеживать четверные и более кратные касания.

Пример: коснитесь и перетащите

Для отслеживания перетаскиваемых объектов вам потребуется подменить три метода: `touchesBegan`, `touchesMoved` и `touchesEnded`. Метод `touchesMoved` будет вызываться каждый раз при перетаскивании объекта по экрану. Когда пользователь отпустит свой палец, метод `touchesEnded` уведомит вас об этом.

Чтобы скомпоновать это приложение, создайте в Xcode приложение DrugDemo на базе представления и добавьте приводимый далее код в ваш класс DrugDemoViewController.m:

```
- (void) touchesBegan:(NSSet *)touches withEvent:(UIEvent *)event
{
    UITouch *touch = [ touches anyObject ];
    CGPoint location = [ touch locationInView: self.view ];
    NSUInteger taps = [ touch tapCount ];

    [ super touchesBegan: touches withEvent: event ];

    NSLog(@"Tap BEGIN at %f, %f Tap count: %d",
          location.x, location.y, taps);
}

- (void) touchesMoved:(NSSet *)touches withEvent:(UIEvent *)event
{
    UITouch *touch = [ touches anyObject ];
    CGPoint oldLocation = [ touch previousLocationInView: self.view ];
    CGPoint location = [ touch locationInView: self.view ];

    [ super touchesMoved: touches withEvent: event ];
    NSLog(@"Finger MOVED from %f, %f to %f, %f",
        oldLocation.x, oldLocation.y, location.x, location.y);
}

- (void) touchesEnded:(NSSet *)touches withEvent:(UIEvent *)event
{
    UITouch *touch = [ touches anyObject ];
    CGPoint location = [ touch locationInView: self.view ];
```

```
    [ super touchesEnded: touches withEvent: event ];
    NSLog(@"Tap ENDED at %f, %f", location.x, location.y);
}
```

После запуска этого приложения вы будете получать уведомления всякий раз, когда пользователь будет опускать или поднимать свой палец, а также когда пользователь будет перемещать свой палец по экрану:

```
BEGIN at 101.000000, 117.953613 Tap count: 1
Finger MOVED from 101.000000, 117.953613 to 102.000000, 117.953613
Finger MOVED from 102.000000, 117.953613 to 104.000000, 117.953613
Finger MOVED from 104.000000, 117.953613 to 105.000000, 117.953613
Finger MOVED from 105.000000, 117.953613 to 107.000000, 117.953613
Finger MOVED from 107.000000, 117.953613 to 109.000000, 116.953613
Finger MOVED from 109.000000, 116.953613 to 113.000000, 115.953613
Finger MOVED from 113.000000, 115.953613 to 116.000000, 115.953613
Finger MOVED from 116.000000, 115.953613 to 120.000000, 114.953613
Finger MOVED from 120.000000, 114.953613 to 122.000000, 114.953613
...
Tap ENDED at 126.000000, 144.953613
```

Поскольку по мере перемещения пользователем пальца вы будете получать несколько различных уведомлений, то важно убедиться в том, что ваше приложение не выполняет никаких функций, интенсивно загружающих ресурсы, до тех пор, пока пользователь не отпустит свой палец, или до момента истечения заданных извне временных интервалов. Например, если ваше приложение является игрой, то метод `touchesMoved` — вряд ли самое подходящее место для записи перемещения символов. Вместо этого используйте данный метод для помещения в очередь этого перемещения и примените отдельный механизм, например, `NSTimer`, для выполнения действительных перемещений. В противном случае большие перемещения мыши будут замедлять ваше приложение, поскольку оно будет пытаться обрабатывать слишком много событий в столь короткий промежуток времени.

Обработка множественного касания

По умолчанию представление настроено на получение только событий прикосновений одного пальца. Чтобы разрешить представлению обрабатывать жесты множественных касаний, задайте метод `multipleTpuchEnabled`. В пре-

дыдущем примере это сделает подмена метода viewDidLoad представлений, как показано далее:

```
- (void)viewDidLoad {
    self.view.multipleTouchEnabled = YES;
    [ super viewDidLoad ];
}
```

После этого события прикосновений как одного пальца, так и нескольких будут отправляться методам уведомлений о касаниях. Вы можете определить количество касаний на экране, взглянув на число событий UITouch, предоставляемое NSSet из первого аргумента. Во время жеста каждому из методов касаний предоставляются два события UITouch, по одному для каждого пальца. Вы можете определить это, выполнив простейший подсчет предоставляемых объектов NSSet:

```
int fingerCount = [ touches count ];
```

Чтобы определить количество использованных в жесте пальцев, запросите UIEvent. Это позволит вам определить то, когда был отпущен последний палец в жесте:

```
int fingersInGesture = [ [ event touchesForView: self.view ] count ];
if (fingerCount == fingersInGesture) {
    NSLog(@"All fingers present in the event");
}
```

Отслеживание пинчей: PinchMe

Этот пример похож на предыдущие примеры этой главы, но вместо отслеживания перемещения одного пальца отслеживается перемещение обоих пальцев для определения фактора масштабирования при операции пинча (pinch). Затем вы сможете легко применить фактор масштабирования для изменения размера изображения или выполнения аналогичных операций с помощью пинча. Как и раньше, в вашем контроллере представлений вы подмените три метода: touchesBegan, touchesMoved и touchesEnded. Чтобы разрешить жесты множественных касаний, вы подмените еще и метод viewDidLoad. Метод touchesMoved будет вызываться каждый раз при перетаскивании объекта мышью по экрану, это время будет рассчитываться фактор масштабирования. Когда пользователь будет отпускать свой палец, метод touchesEnded уведомит вас об этом.

Чтобы скомпоновать это приложение, создайте в Xcode приложение PinchMe на базе представления и добавьте код из листинга 4.1 в ваш класс PinchMeViewController.m.

Листинг 4.1. Код для PinchMeViewController.m

```
- (void)viewDidLoad {
    [super viewDidLoad];
    [ self.view setMultipleTouchEnabled: YES ];
}

- (void) touchesBegan:(NSSet *)touches withEvent:(UIEvent *)event
{
    UITouch *touch = [ touches anyObject ];
    CGPoint location = [ touch locationInView: self.view ];
    NSUInteger taps = [ touch tapCount ];

    [ super touchesBegan: touches withEvent: event ];

    NSLog(@"Tap BEGIN at %f, %f Tap count: %d",
          location.x, location.y, taps);
}

- (void) touchesMoved:(NSSet *)touches withEvent:(UIEvent *)event
{
    int finger = 0;
    NSEnumerator *enumerator = [ touches objectEnumerator ];
    UITouch *touch;
    CGPoint location[2];
    while ((touch = [ enumerator nextObject ]) && finger < 2)
    {
        location[finger] = [ touch locationInView: self.view ];
        NSLog(@"Finger %d moved: %fx%f",
              finger+1, location[finger].x, location[finger].y);
        finger++;
    }
```

```
    if (finger == 2) {
        CGPoint scale;
        scale.x = fabs(location[0].x - location[1].x);
        scale.y = fabs(location[0].y - location[1].y);
        NSLog(@"Scaling: %.0f x %.0f", scale.x, scale.y);
    }
    [ super touchesMoved: touches withEvent: event ];
}

- (void) touchesEnded:(NSSet *)touches withEvent:(UIEvent *)event
{
    UITouch *touch = [ touches anyObject ];
    CGPoint location = [ touch locationInView: self.view ];

    [ super touchesEnded: touches withEvent: event ];
    NSLog(@"Tap ENDED at %f, %f", location.x, location.y);
}
```

Во время выполнения приложения откройте консоль. Удерживая вариантную клавишу (option key), вы сможете в симуляторе iPhone симулировать жест пинча. Во время перемещения мыши вы увидите записи координат для обоих пальцев и x/y-разницы между ними:

```
Untitled[5039:20b] Finger 1 moved: 110.000000x293.046387
Untitled[5039:20b] Finger 2 moved: 210.000000x146.953613
Untitled[5039:20b] Scaling: 100 . 146
Untitled[5039:20b] Finger 1 moved: 111.000000x289.046387
Untitled[5039:20b] Finger 2 moved: 209.000000x150.953613
Untitled[5039:20b] Scaling: 98 . 138
Untitled[5039:20b] Finger 1 moved: 112.000000x285.046387
Untitled[5039:20b] Finger 2 moved: 208.000000x154.953613
Untitled[5039:20b] Scaling: 96 . 130
Untitled[5039:20b] Finger 1 moved: 113.000000x282.046387
Untitled[5039:20b] Finger 2 moved: 207.000000x157.953613
Untitled[5039:20b] Scaling: 94 . 124
Untitled[5039:20b] Finger 1 moved: 113.000000x281.046387
Untitled[5039:20b] Finger 2 moved: 207.000000x158.953613
Untitled[5039:20b] Scaling: 94 . 122
```

Отслеживание перетаскивания значков: TouchDemo

Для наиболее полного примера использования API касаний пример TouchDemo демонстрирует, как вы можете использовать API касаний для отслеживания перемещений по экрану отдельных пальцев. С Web-узла автоматически загружаются и отображаются на экране четыре демонстрационных изображения. Пользователь может использовать один или несколько пальцев для одновременного перетаскивания значков на новые места. Данный пример отслеживает каждое касание отдельно; он изменяет положение каждого значка по мере перетаскивания его пользователем (рис. 4.1).

Вы можете скомпилировать это приложение, показанное в листингах 4.2—4.6, с помощью SDK, создав проект приложения TouchDemo на базе представления. Если вы хотите увидеть, как создавать все эти объекты с нуля, то удалите код Interface Builder.

Рис. 4.1. Пример TouchDemo

Листинг 4.2. Прототипы делегата приложения TouchDemo (TouchDemoAppDelegate.h)

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

@class TouchDemoViewController;
```

```
@interface TouchDemoAppDelegate : NSObject <UIApplicationDelegate> {
    UIWindow *window;
    TouchDemoViewController *viewController;
}

@property (nonatomic, retain) IBOutlet UIWindow *window;
@property (nonatomic, retain) IBOutlet TouchDemoViewController
*viewController;

@end
```

Листинг 4.3. Делегат приложения TouchDemo (TouchDemoAppDelegate.m)

```
#import "TouchDemoAppDelegate.h"
#import "TouchDemoViewController.h"

@implementation TouchDemoAppDelegate

@synthesize window;
@synthesize viewController;

- (void)applicationDidFinishLaunching:(UIApplication *)application {
    CGRect screenBounds = [ [ UIScreen mainScreen ] applicationFrame ];

    self.window = [ [ [ UIWindow alloc ] initWithFrame: screenBounds ]
        autorelease
    ];

    viewController = [ [ TouchDemoViewController alloc ] init ];

    [ window addSubview: viewController.view ];
    [ window makeKeyAndVisible ];
}

- (void)dealloc {
    [ viewController release ];
```

```
    [ window release ];
    [ super dealloc ];
}

@end
```

Листинг 4.4. Прототипы контроллера представлений TouchDemo (TouchDemoViewController.h)

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

@interface TouchView : UIView {
    UIImage *images[4];
    CGPoint offsets[4];
    int tracking[4];
}

@end

@interface TouchDemoViewController : UIViewController {
    TouchView *touchView;
}

@end
```

Листинг 4.5. Контроллер представлений TouchDemo (TouchDemoViewController.m)

```
#import "TouchDemoViewController.h"

@implementation TouchView

- (id)initWithFrame:(CGRect)frame {
    frame.origin.y = 0.0;
    self = [ super initWithFrame: frame ];
    if (self != nil) {
      self.multipleTouchEnabled = YES;
```

```
      for (int i=0; i<4; i++) {
        NSURL *url = [ NSURL URLWithString: [
          NSString stringWithFormat:
            @"http://www.zdziarski.com/demo/%d.png", i+1 ]
        ];
        images[i] = [ [ UIImage alloc ]
           initWithData: [ NSData dataWithContentsOfURL: url ]
        ];
        offsets[i] = CGPointMake(0.0, 0.0);
      }

      offsets[0] = CGPointMake(0.0, 0.0);
      offsets[1] = CGPointMake(
            self.frame.size.width - images[1].size.width, 0.0);
      offsets[2] = CGPointMake(0.0,
            self.frame.size.height - images[2].size.height);
      offsets[3] = CGPointMake(
            self.frame.size.width  - images[3].size.width,
            self.frame.size.height - images[3].size.height);
    }
    return self;
}

- (void)drawRect:(CGRect)rect {
    for(int i=0; i<4; i++ ) {
        [ images[i] drawInRect: CGRectMake(offsets[i].x, offsets[i].y,
            images[i].size.width, images[i].size.height)
        ];
    }
}

- (void)touchesBegan:(NSSet *)touches withEvent:(UIEvent *)event {
    UITouch *touch;
    int touchId = 0;

    NSEnumerator *enumerator = [ touches objectEnumerator ];
    while ((touch = (UITouch *) [ enumerator nextObject ])) {
```

```
      tracking[touchId] = -1;
      CGPoint location = [ touch locationInView: self ];
      for (int i=3;i>=0;i--) {
        CGRect rect = CGRectMake(offsets[i].x, offsets[i].y,
                images[i].size.width, images[i].size.height);
        if (CGRectContainsPoint(rect, location)) {
          NSLog(@"Begin Touch ID %d Tracking with image %d\n",
                touchId, i);
          tracking[touchId] = i;
          break;
        }
      }
      touchId++;
    }
}

- (void)touchesMoved:(NSSet *)touches withEvent:(UIEvent *)event {
    UITouch *touch;
    int touchId = 0;
    NSEnumerator *enumerator = [ touches objectEnumerator ];
    while ((touch = (UITouch *) [ enumerator nextObject ])) {
      if (tracking[touchId] != -1) {
        NSLog(@"Touch ID %d Tracking with image %d\n",
              touchId, tracking[touchId]);

        CGPoint location = [ touch locationInView: self ];
        CGPoint oldLocation = [ touch previousLocationInView: self ];
        offsets[tracking[touchId]].x += (location.x - oldLocation.x);
        offsets[tracking[touchId]].y += (location.y - oldLocation.y);

        if (offsets[tracking[touchId]].x < 0.0)
            offsets[tracking[touchId]].x = 0.0;
        if (offsets[tracking[touchId]].x +
            images[tracking[touchId]].size.width > self.frame.size.width)
        {
          offsets[tracking[touchId]].x = self.frame.size.width
              - images[tracking[touchId]].size.width;
        }
```

```
        if (offsets[tracking[touchId]].y < 0.0)
            offsets[tracking[touchId]].y = 0.0;
        if (offsets[tracking[touchId]].y +
            images[tracking[touchId]].size.height >
              self.frame.size.height)
        {
           offsets[tracking[touchId]].y = self.frame.size.height
                 - images[tracking[touchId]].size.height;
        }
      }
      touchId++;
    }

    [ self setNeedsDisplay ];
}
@end

@implementation TouchDemoViewController

- (void)loadView {
    [ super loadView ];

    touchView = [ [ TouchView alloc ] initWithFrame: [
        [ UIScreen mainScreen ] applicationFrame ]
    ];
    self.view = touchView;
}

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

@end
```

Листинг 4.6. Функция `main` для TouchDemo (main.m)

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

int main(int argc, char *argv[]) {

    NSAutoreleasePool * pool = [ [ NSAutoreleasePool alloc ] init ];
    int retVal = UIApplicationMain(argc, argv, nil,
                                   @"TouchDemoAppDelegate");
    [ pool release ];
    return retVal;
}
```

Как это работает

Пример TouchDemo работает следующим образом:

1. Когда приложение запускается, оно порождает новое окно и контроллер представлений. Создается `UIView`-подкласс под названием `TouchView` и прикрепляется к контроллеру представлений.
2. Когда пользователь опускает один или несколько пальцев, уведомляется метод `touchesBegun`. Он сравнивает положение каждого касания с началом координат и размером изображения, чтобы определить, какого из изображений коснулся пользователь. Он записывает индекс в массив `tracking`.
3. Когда пользователь перемещает палец, уведомляется метод `touchesMoved`. Он определяет, какого из изображений изначально коснулся пользователь, и изменяет на экране его начало координат на соответствующее количество. Затем он вызывает метод `setNeedDisplay`, который вызывает метод `drawRect` для перерисовки содержимого представления.

Для дальнейшего изучения

В заголовочных файлах вашего SDK проверьте наличие следующих прототипов: UITouch.h и UIEvent.h. Вы найдете их в папке /Developer/Platforms/iPhoneOS.platform внутри каталога Headers библиотеки UI Kit.

Глава 5

Программирование уровней с использованием Quartz Core

Библиотека Quartz Core на настольных системах Leopard называется Core Animation. Quartz Core предоставляет базовые классы для управления уровнями объектов `UIView`. Она также используется при создании 3D-преобразований 2D-объектов для анимации и других графических эффектов.

Чтобы воспользоваться библиотекой Quartz Core, вам потребуется добавить ее в ваш проект Xcode. Щелкните правой кнопкой мыши по папке Frameworks в вашем проекте, а затем в появившемся контекстном меню выберите команду **Add Framework**. Перейдите к папке QuartzCore.framework, а затем нажмите кнопку **Add**.

> ***Внимание!***
>
> Чтобы найти библиотеку Quartz Core, вам, возможно, придется вручную перейти либо в папку /Developer/Platforms/iPhoneOS.platform, либо в папку /Developer/Platforms/iPhoneSimulator.platform и найти папку Frameworks в вашем SDK.

Понятие уровней

Уровень[1] (layer) — это объект низкого уровня, имеющийся во многих визуализируемых классах. Уровни аналогичны листам доски объявлений, которой

[1] Layer (англ.) — уровень, слой.

принадлежит содержимое объекта. Уровень ведет себя как гибкая подложка для отображаемого содержимого объекта и может сгибать или скручивать объект на экране несколькими способами. Каждый объект, который может визуализировать себя, а таковыми являются объекты, порожденные от класса `UIView`, имеет, по крайней мере, один уровень, к которому привязано его содержимое.

Например, класс `UIImageView` содержит всю основную информацию о двумерном изображении: область его отображения, разрешение и различные методы для работы с изображением и его визуализации. Само изображение привязано к уровню родительского класса `UIView`, который похож на подложку картины. Большинство основных уровней ведет себя как единый лист картона и всего лишь отображает изображение, как оно есть. Кроме того, уровни могут содержать несколько подуровней, которые рассматриваются как диапозитивы, каждый со своим содержимым, сложенные друг на друга для создания одного составного изображения.

Будучи гибкими, уровни могут использоваться для управления принадлежащим им содержимым. 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. Уровень можно преобразовывать, добавлять анимацию или же поворачивать, наклонять и масштабировать изображение. 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 воспринимает ту форму, которую получил уровень. Уровни могут преобразовывать не только просто изображения. Они отображают вывод для всех других классов представлений, рассматриваемых в *главах 3* и *10*, также размещающихся наверху уровня.

Чтобы получить доступ к уровню, считайте свойство `layer` базового класса `UIView`:

```
CALayer *layer = myView.layer;
```

Все объекты, порожденные от `UIView`, наследуют это свойство, т. е. вы можете преобразовывать, масштабировать, поворачивать и даже анимировать панели навигации, таблицы, текстовые поля и другие типы классов представлений.

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.

Иерархия уровней

Уровни обладают целым рядом методов и свойств общего назначения для работы с подуровнями и визуализации. Эти методы позволяют вам "собирать" несколько отдельных уровней экрана воедино для формирования составного экранного изображения.

Один уровень может иметь много подуровней. При формировании финального экрана подуровни упорядочиваются и сшиваются воедино. Рассмотрим пример игры-стрелялки. Ваша игра может состоять из одного уровня для формирования персонажей игры, одного уровня для формирования фона и одного уровня для формирования микродисплея для чтения на ходу (head-up display — HUD). Для каждого из них у вас может иметься класс `UIView`, и четвертый класс, представляющий экран игры:

```
UIView *gameView = [ [ UIView alloc ] initWithFrame:
    [ [ UIScreen mainScreen ] applicationFrame ]
];

UIView *backgroundView = [ [ UIView alloc ] initWithFrame...
UIView *characterView = [ [ UIView alloc ] initWithFrame...
UIView *HUDView = [ [ UIView alloc ] initWithFrame...
```

Вы можете прикрепить каждый из трех уровней класса `UIView` к объекту `gameView` с помощью метода `addSublayer` класса `CALayer`:

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

CALayer *gameLayer = gameView.layer;
[ gameLayer addSublayer: backgroundView.layer ];
[ gameLayer addSublayer: characterView.layer ];
[ gameLayer addSublayer: HUDView.layer ];
```

Когда объект `gameView` отображается на экране, все три подуровня объединяются и визуализируются. Каждый класс формирует собственный подуровень отдельно, но когда уровни игры выводятся на экран, все три уровня объединяются.

Уровень `gameView` — не единственный, который вы можете добавить. Подуровни сами по себе могут иметь свои подуровни, и может быть построена целая иерархия уровней. Например, ваша игра может добавить уровень к

уровню `HUDView` для отображения одного компонента HUD, например, логотипа команды разработчиков:

```
UIImageView *teamLogoView = [ [ UIImageView alloc ] initWithImage:
    myTeamLogo ];
[ HUDView.layer addSublayer: teamLogoView.layer ];
```

Всякий раз при визуализации уровня `HUDView` все его подуровни также будут визуализироваться.

Размер и смещение

Чтобы изменить размер или начало координат, в котором отображается уровень, установите свойство уровня `frame`. Фрейм уровня ведет себя таким же образом, что и фрейм окна или класса представления: принимая координаты X и Y в качестве начала координат и размер области. Приведенный далее пример устанавливает начало координат уровня `teamLogoView` в точке (150, 100) и его размер, равным 50×75:

```
CGRect teamLogoFrame = CGRectMake(150.0, 100.0, 50.0, 75.0);
teamLogoView.layer.frame = teamLogoFrame;
```

Иначе вы можете задать расположение уровня, не изменяя его размер, установив свойство уровня `position`. Это свойство принимает структуру `CGPoint`, определяя смещение для отображения уровня на экране. В отличие от свойства `frame` свойство `position` задает середину уровня, а не левый верхний угол:

```
CGPoint logoPosition = CGPointMake(75.0, 50.0);
teamLogoView.layer.position = logoPosition;
```

Упорядочивание и отображение

Помимо добавления подуровней класс `CALayer` предоставляет целый ряд различных методов для вставки, упорядочивания и удаления уровней.

Когда вы добавляете уровень с помощью `addSublayer`, то он добавляется на самый верх иерархии уровней, поэтому он будет отображаться впереди всех остальных существующих подуровней. Используя множество альтернативных методов под названием `insertSublayer`, вы можете вставить новые уровни между существующими уровнями.

Чтобы вставить уровень в определенное место, воспользуйтесь аргументом `atIndex`:

```
[ gameLayer insertSublayer: HUDView.layer atIndex: 1 ];
```

Чтобы вставить уровень выше или ниже другого уровня, добавьте аргумент `above` или `below`:

```
[ gameLayer insertSublayer: HUDView.layer below: backgroundView.layer ];
[ gameLayer insertSublayer: HUDView.layer above: characterView.layer ];
```

Чтобы удалить уровень из его родительского (супер) уровня, вызовите метод `removeFromSuperlayer` дочернего уровня:

```
[ HUDView.layer removeFromSuperlayer ];
```

Чтобы заменить существующий подуровень другим уровнем, воспользуйтесь методом `replaceSunlayer`:

```
[ gameLayer replaceSublayer: backgroundView.layer with: newBackground-
View.layer ];
```

Чтобы сохранить подуровень в стеке, но сделать его невидимым, задайте свойство уровня `hidden`. Вы можете использовать приведенный далее код для включения и отключения отображения HUD, не прибегая к действительному удалению уровня:

```
- (void) ToggleHUD {
    HUDView.layer.hidden = (HUDView.layer.hidden == NO) ? YES : NO;
}
```

Визуализация

При обновлении уровня все изменения отображаются на экране не сразу. Это позволяет вам производить запись в уровень, не показывая все пользователю до тех пор, пока не будут завершены все операции. Когда уровень будет готов к перерисовке, вызовите метод уровня `setNeedsDisplay`:

```
[ gameLayer setNeedsDisplay ];
```

Иногда может возникнуть необходимость в перерисовке части содержимого уровня. Перерисовка всего экрана может вызывать снижение производительности. Чтобы перерисовать только часть экрана, нуждающуюся в обновлении, воспользуйтесь методом `setNeedsDisplayInRect`, передав в него структуру `CGRect` области обновления:

```
CGRect teamLogoFrame = CGRectMake(150.0, 100.0, 50.0, 75.0);
[ gameLayer setNeedsDisplayInRect: teamLogoFrame ];
```

Если для формирования изображений вы используете библиотеку Core Graphics, то вы можете визуализировать напрямую в контекст Core Graphics. Для этого воспользуйтесь методом `renderInContext`:

```
CGContextRef myCGContext = UIGraphicsGetCurrentContext();
[ gameLayer renderInContext: myCGContext ];
```

Преобразования

Чтобы добавить к уровню 3D- или аффинное преобразование, установите свойство уровня `transform` или `affine` соответственно. Более подробно о преобразованиях вы узнаете далее в этой главе:

```
characterView.layer.transform = CATransform3DMakeScale(-1.0, -1.0, 1.0);

CGAffineTransform transform = CGAffineTransformMakeRotation(45.0);
backgroundView.layer.affineTransform = transform;
```

Анимация уровней

Возможности библиотеки Quartz Core гораздо шире, чем ее возможности как просто склеивающего уровня. Quartz Core может использоваться при трансформации объекта 2D в текстуру 3D для создания великолепных переходов между представлениями.

В *главе 3* рассматривались переходы как средства перехода между различными объектами `UIView`. Вы применяли переходы и анимации напрямую к уровням или подуровням. Как вы помните из *главы 3*, анимации создаются как объекты `CATransition`.

Переходы уровней дополняют существующий класс `CATransition`, предоставляя способ добавления анимации с помощью анимационных средств Quartz Core. Это позволяет разработчику использовать преимущества 3D-возможностей, предлагаемых Quartz Core, не внося при этом существенных изменений в свой код. Когда анимируется уровень, к уровню прикрепляется объект `CATransition` или объект `CAAnimation`. Затем этот уровень вызывает Quartz Core для порождения нового потока, который берет на себя всю графическую обработку анимации. Разработчику необходимо только добавить желаемую анимацию для доработки существующего уровня.

Переход можно создать с помощью следующего фрагмента кода:

```
CATransition *animation = [[CATransition alloc] init];
animation.duration = 1.0;
animation.timingFunction = [ CAMediaTimingFunction
    functionWithName: kCAMediaTimingFunctionEaseIn ];
animation.type = kCATransitionPush;
animation.subtype = kCATransitionFromRight;
```

> ***Внимание!***
>
> На текущий момент количество представленных в iPhone SDK типов переходов, которые может создавать пользователь, сильно ограничено. Однако внутри библиотеки Quartz Core поддерживается около дюжины дополнительных переходов, например, сворачивание страниц и переходы с изменением масштаба, но их использование разрешено только в собственных приложениях Apple.

Вы можете организовать анимацию путем создания объекта `CABasicAnimation`. Приведенный далее пример создает анимацию, поворачивающую уровень на 360°:

```
CABasicAnimation *animation = [ CABasicAnimation
    animationWithKeyPath: @"transform" ];
animation.toValue = [ NSValue valueWithCATransform3D: CATransform3D
MakeRotation(3.1415, 0, 0, 1.0) ];
animation.duration = 2;
animation.cumulative = YES;
animation.repeatCount = 2;
```

После создания анимацию или переход можно применить непосредственно к уровню:

```
[ teamLogoView.layer addAnimation: animation forKey: @"animation" ];
```

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. Набор функций `CATransform3D` позволяет воплощать магические действия под покровом технологии Cover Flow от Apple (описываемой в *гла-*

ве 12) и другие эффекты, используемые в iPhone. iPhone поддерживает масштаб, вращение, аффинные и параллельные преобразования. В данной книге представлено только введение в преобразования уровней, но Core Animation является объемной темой, по которой существует достаточное количество других книг.

> ***Примечание***
>
> Более подробную информацию о создании анимаций можно найти в книге Bill Dudney (Билл Дудней) "Core Animation for Mac OS X and the iPhone" издательства Pragmatic.

Преобразование осуществляется для отдельных уровней, позволяя тем самым нескольким на многоуровневой поверхности преобразованиям происходить одновременно. Платформа Quartz Core выполняет преобразования с помощью объекта `CATransform3D`. Этот объект применяется к уровню представления, 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:

```
CATransform3D myTransform;

myTransform = CATransform3DMakeRotation(angle, x, y, z);
```

Функция `CATransform3DMakeRotation` создает преобразование, которое повернет уровень на количество радиан, определяемое углом `angle`, с использованием осей *x*, *y*, *z*. Значения `x`, `y`, `z` задают оси 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* имеет значение либо –1, либо +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 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);
```

Использованная величина `0.78` представляет радианное значение угла. Чтобы пересчитать градусы в радианы, воспользуйтесь простой формулой: $M\pi/180$. Например, $45\pi/180 = 45 \cdot (3{,}1415)/180 = 0{,}7853$. Если вы собираетесь в своем приложении работать с измерениями в градусах, то можете написать простую функцию преобразования, чтобы ваш код оставался доступным для понимания:

```
double radians(float degrees) {
    return (degrees * 3.14159265) / 180.0;
}
```

После этого вы сможете вызывать эту функцию при создании преобразований:

```
myTransform = CATransform3DMakeRotation(radians(45.0), 0.0, 1.0, 0.0);
```

После создания преобразования примените его к уровню, с которым вы работаете. Объект `CALayer` предоставляет свойство `transform`, используемое для прикрепления преобразования. Уровень будет выполнять любое преобразование, присвоенное данному свойству:

```
imageView.layer.transform = myTransform;
```

При демонстрации объекта он будет отображаться вместе с примененным к нему преобразованием. В своем коде вы все так же будете ссылаться на данный объект как на объект 2D, но выводиться на экран он будет согласно созданному преобразованию.

Развлечение с уровнями: BounceDemo

В данном примере создаются два уровня из загруженных из Интернета изображений. Затем эти уровни прикрепляются к контроллеру представлений, где ими можно легко манипулировать с помощью таймера. Этот таймер определяет расположение на экране каждого уровня и добавляет анимацию. Представленный код может служить хорошим функциональным руководством для создания ваших собственных процедур работы с уровнями (рис. 5.1).

Вы можете скомпилировать это приложение, показанное в листингах 5.1—5.5, с помощью SDK, создав проект приложения BounceDemo на базе пред-

ставления. Если вы хотите увидеть, как создавать все эти объекты с нуля, то удалите код Interface Builder.

Рис. 5.1. Пример BounceDemo

Листинг 5.1. Прототипы делегата приложения BounceDemo (BounceDemoAppDelegate.h)

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

@class BounceDemoViewController;

@interface BounceDemoAppDelegate : NSObject <UIApplicationDelegate> {
    UIWindow *window;
    BounceDemoViewController *viewController;
}

@property (nonatomic, retain) IBOutlet UIWindow *window;
@property (nonatomic, retain) IBOutlet BounceDemoViewController
   *viewController;

@end
```

Листинг 5.2. Делегат приложения BounceDemo (BounceDemoAppDelegate.m)

```
#import "BounceDemoAppDelegate.h"
#import "BounceDemoViewController.h"

@implementation BounceDemoAppDelegate

@synthesize window;
@synthesize viewController;

- (void)applicationDidFinishLaunching:(UIApplication *)application {
    CGRect screenBounds = [ [ UIScreen mainScreen ] applicationFrame ];

    self.window = [ [ [ UIWindow alloc ] initWithFrame: screenBounds ]
        autorelease
    ];

    viewController = [ [ BounceDemoViewController alloc ] init ];

    [ window addSubview: viewController.view ];
    [ window makeKeyAndVisible ];
}

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

@end
```

Листинг 5.3. Прототипы контроллера представлений BounceDemo (BounceDemoViewController.h)

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

@interface BounceDemoViewController : UIViewController {
    UIImageView *image1, *image2;
```

```
    CGPoint directionImage1, directionImage2;
    NSTimer *timer;
}
- (void) handleTimer: (NSTimer *) timer;
@end
```

Листинг 5.4. Контроллер представлений BounceDemo (BounceDemoViewController.m)

```
#import "BounceDemoViewController.h"
#import <QuartzCore/QuartzCore.h>

@implementation BounceDemoViewController

- (id)init {
    self = [ super init ];
    if (self != nil) {
      UIImage *image;
      NSURL *url;

      url = [ NSURL URLWithString:
             @"http://www.zdziarski.com/demo/1.png" ];
      image = [ UIImage imageWithData:
                [ NSData dataWithContentsOfURL: url ] ];
      image1 = [ [ UIImageView alloc ] initWithImage: image ];
      directionImage1 = CGPointMake(-1.0, -1.0);
      image1.layer.position = CGPointMake((image.size.width / 2)+1,
                                          (image.size.width / 2)+1);

      url = [ NSURL URLWithString:
               @"http://www.zdziarski.com/demo/2s.png" ];
      image = [ UIImage imageWithData:
                [ NSData dataWithContentsOfURL: url ] ];
      image2 = [ [ UIImageView alloc ] initWithImage: image ];
      directionImage2 = CGPointMake(1.0, 1.0);
      image2.layer.position = CGPointMake((image.size.width / 2)+1,
                                          (image.size.width / 2)+1);
```

```
      [ self.view.layer addSublayer: image2.layer ];
      [ self.view.layer addSublayer: image1.layer ];
    }
    return self;
}

- (void)loadView {
    [ super loadView ];
}

- (void)viewDidLoad {
    [ super viewDidLoad ];

    timer = [ NSTimer scheduledTimerWithTimeInterval: 0.01
                                        target: self
                                      selector: @selector(handleTimer:)
                                        erInfo: nil
                                        epeats: YES
            ];
    CABasicAnimation *anim1 =
        [ CABasicAnimation animationWithKeyPath: @"transform" ];
    anim1.toValue = [ NSValue valueWithCATransform3D:
        CATransform3DMakeRotation(3.1415, 1.0, 0, 0)
    ];

    anim1.duration = 2;
    anim1.cumulative = YES;
    anim1.repeatCount = 1000;
    [ image1.layer addAnimation: anim1 forKey: @"transformAnimation" ];
}

- (void)didReceiveMemoryWarning {
    [ super didReceiveMemoryWarning ];
}

- (void)dealloc {
    [ timer invalidate ];
```

```
    [ image1 release ];
    [ image2 release ];
    [ super dealloc ];
}

- (void) handleTimer: (NSTimer *) timer {
    CGSize size;
    CGPoint origin;

    /* Перемещаем image1 */
    size = [ image1 image ].size;
    if (image1.layer.position.x <=
        ((size.width / 2) + 1) - self.view.frame.origin.x)
        directionImage1.x = 1.0;
    if (image1.layer.position.x + (size.width / 2) + 1 >=
        (self.view.frame.size.width - self.view.frame.origin.x) - 1)
        directionImage1.x = -1.0;
    if (image1.layer.position.y <=
        ((size.height / 2) + 1) - self.view.frame.origin.y)
        directionImage1.y = 1.0;
    if (image1.layer.position.y + (size.height / 2) + 1 >=
        (self.view.frame.size.height - self.view.frame.origin.y) - 1)
        directionImage1.y = -1.0;
    origin = image1.layer.position;
    origin.x += directionImage1.x;
    origin.y += directionImage1.y;
    image1.layer.position = origin;

    /* Перемещаем image2 */
    size = [ image2 image ].size;
    if (image2.layer.position.x <=
        ((size.width / 2) + 1) - self.view.frame.origin.x)
        directionImage2.x = 1.0;
    if (image2.layer.position.x + (size.width / 2) + 1
        >= (self.view.frame.size.width - self.view.frame.origin.x) - 1)
        directionImage2.x = -1.0;
```

```
    if (image2.layer.position.y <=
        ((size.height / 2) + 1) - self.view.frame.origin.y)
        directionImage2.y = 1.0;
    if (image2.layer.position.y + (size.height / 2) + 1 >=
        (self.view.frame.size.height - self.view.frame.origin.y) - 1)
        directionImage2.y = -1.0;
    origin = image2.layer.position;
    origin.x += directionImage2.x;
    origin.y += directionImage2.y;
    image2.layer.position = origin;

    [ self.view setNeedsDisplayInRect: image1.layer.frame ];
    [ self.view setNeedsDisplayInRect: image2.layer.frame ];
}
@end
```

Листинг 5.5. Функция `main` для BouceDemo (main.m)

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

int main(int argc, char *argv[]) {
    NSAutoreleasePool * pool = [ [ NSAutoreleasePool alloc ] init ];
    int retVal = UIApplicationMain(argc, argv, nil,
                                   @"BounceDemoAppDelegate");
    [ pool release ];
    return retVal;
}
```

Как это работает

Вот как работает пример BouceDemo:

1. При загрузке приложения уведомляется его класс `BouceDemoAppDelegate` посредством его метода `applicationDidFinishLaunching`. Этот метод создает окно и порождает контроллер представлений.
2. Метод `init` контроллера представлений создает два объекта `UIImageView`, загружая изображения из Интернета. Уровни для этих объ-

ектов добавляются к главному представлению контроллера представлений как подуровни.

3. При загрузке представления вызывается метод `viewDidLoad`, который создает таймер. Метод `handleTimer` определяет расположение каждого уровня по одному пикселу за каждый свой вызов, создавая эффект подергивания изображения на экране. Метод `viewDidLoad` также создает две анимации: одну для вертикального транспонирования уровня, а другую — для горизонтального транспонирования уровня. Обе анимации добавляются к целевым объектам и запускаются автоматически.

Для дальнейшего изучения

Теперь, когда у вас появилось представление о работе с уровнями и анимацией, прежде чем продолжить работу попробуйте кое-что сделать.

- Изучите статью Core Animation на Web-узле Apple Developer Connection. Измените анимации в проекте, применив эффекты постепенного исчезновения, масштабирования и другие анимации.
- В заголовочных файлах вашего SDK проверьте наличие следующих прототипов: CALayer.h, CAAnimation.h и CATransform3D.h. Вы найдете их в папке /Developer/Platforms/iPhoneOS.platform внутри каталога Headers библиотеки Quartz Core.

Глава 6

Создание шума: Audio Toolbox и AVFoundation

Библиотека AVFoundation была представлена в iPhone SDK версии 2.2 и предоставляла функциональность для воспроизведения и микширования звуковых файлов, а также базового управления звуком. Если вы в свое приложение добавляете простейшие звуки, то с помощью этой библиотеки можете это сделать быстро и безболезненно.

Библиотека Audio Toolbox отвечает за управление выводом цифрового звука на iPhone. В отличие от многих описываемых в этой книге библиотек данная библиотека является наиболее C-ориентированной. Эта мощная библиотека позволяет вам создавать и записывать цифровой звук. Помимо приводимых в этой главе примеров, для Audio Toolbox написано много руководств, которые доступны на Web-узле Apple Developer Connection. Таковыми являются:

- Core Audio Overview: AudioToolbox.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 библиотека Audio Toolbox существует и для настольной платформы, то она достаточно хорошо документирована. Здесь мы не будем детально описывать ее, а уделим внимание только тем ее частям, которые относятся к iPhone SDK. Многие части этой библиотеки, например, MIDI-контроллеры и Music Player API, не имеют никакого отношения к iPhone или недоступны для него.

Чтобы воспользоваться библиотекой Audio Toolbox, вам потребуется добавить ее в ваш проект Xcode. Для этого щелкните правой кнопкой мыши по папке Frameworks в вашем проекте Xcode, а затем в появившемся контекстном меню выберите команду **Add | Existing Frameworks**. Перейдите к папке iPhone SDK Framework и выберите папку AudioToolbox.framework.

Библиотека AVFoundation

В iPhone SDK версии 2.2 была представлена новая библиотека для воспроизведения и микширования звука: AVFoundation. Эта библиотека предоставляет легкий в применении интерфейс для воспроизведения простых звуковых файлов. Она может показаться простым классом, однако содержит некоторые хитрости. Эта библиотека не только воспроизводит отдельные звуковые файлы, но может также осуществлять одновременное воспроизведение нескольких звуков. Класс `AVAudioPlayer` предоставляет возможность управления уровнем громкости для каждого сэмпла (sample), позволяя тем самым вам выполнять простейшее микширование звуков, а также может воспроизводить целый ряд различных форматов файлов, включая MP3, M4A, WAV, CAF, AIF и др. Помимо этого, он предоставляет свойства для чтения уровней мощности, позволяя тем самым делать то, что обожает делать большинство фанатов с момента появления Apple, — создавать собственные волюметры (VU meters).

Библиотека AVFoundation оказывается весьма полезной при добавлении поддержки звука в такие приложения, как приложения обмена мгновенными сообщениями или несложные игры, которым не требуется сложный цифровой звук. Чтобы создать потоки цифрового звука, вам все-таки потребуется библиотека Audio Toolbox, описываемая далее в этой главе.

Чтобы воспользоваться библиотекой AVFoundation, вам нужно добавить ее в ваш проект. Для этого щелкните правой кнопкой мыши по папке Frameworks в вашем проекте Xcode, а затем в появившемся контекстном меню выберите команду **Add | Existing Frameworks**. Перейдите к папке iPhone SDK Framework и выберите папку AVFoundation.framework.

> ***Внимание!***
>
> Чтобы воспользоваться библиотекой AVFoundation, вам потребуется iPhone SDK версии не ниже 2.2.

Звуковой проигрыватель

Класс `AVAudioPlayer` инкапсулирует единичный звук для воспроизведения. Проигрыватель инициализируется с объектом `NSURL`, указывающим на ресурс для воспроизведения. Чтобы одновременно воспроизвести несколько звуков, вы можете создать новый объект `AVAudioPlayer` для каждого звука. Рассматривайте звуковой проигрыватель как одну звуковую дорожку (track) в рамках мультитрекового микшерного пульта:

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

NSError *err;
AVAudioPlayer *player = [ [ AVAudioPlayer alloc ]
  initWithContentsOfURL: [ NSURL fileURLWithPath:
    [ [ NSBundle mainBundle ] pathForResource: @"sample"
      ofType:@"m4a" inDirectory:@"/" ] ]
  error: &err
];
```

Вы также можете инициализировать проигрыватель с объектом `NSData`, указывающим на необработанные данные для вашего звука, размещенные в памяти:

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

NSError *err;
AVAudioPlayer *player = [ [ AVAudioPlayer alloc ]
  initWithData: myData
  error: &err
];
```

Свойства проигрывателя

После создания и инициализации объекта `AVAudioPlayer` вы можете задать для него различные свойства.

С помощью свойства `volume` вы можете микшировать выход от нескольких проигрывателей. Уровень громкости представляется значением между 0,0 и 1,0:

```
player.volume = 0.5;
```

Если звук должен повторяться, то установите свойство `numberOfLoops` в ненулевое значение, соответствующее количеству повторений. По умолчанию звук воспроизводится только один раз:

```
player.numberOfLoops = 3;
```

Если вы хотите, чтобы сэмпл воспроизводился с определенной временной задержкой, то можете поместить "головку воспроизведения" в нужное вам место, задав свойство `currentTime`. В дальнейшем вы сможете считать эту задержку во время воспроизведения сэмпла. Это свойство использует `NSTimeInterval` — значение с плавающей точкой двойной точности (double floating-point), представляющее количество секунд в сэмпле для воспроизведения:

```
NSTimeInterval currentTime = player.currentTime;
```

```
player.currentTime = 5.0;
```

Помимо свойств, которые вы можете настроить вручную, существует еще несколько свойств только для чтения, которые могут помочь определить характеристики загружаемого вами звука.

Из свойства `numberOfChannels` вы можете прочитать количество каналов, имеющихся в сэмпле. Оно возвращает один канал для моносэмплов и два канала для стереосэмплов:

```
NSUInteger channels = player.numberOfChannels
```

Также вы можете считать длительность сэмпла (в секундах) посредством свойства `duration`. Оно тоже представлено как `NSTimeInterval`, имеющий тип с плавающей точкой двойной точности:

```
NSTimeInterval duration = player.duration
```

Воспроизведение звуков

Если вы помещаете в очередь звук и хотите иметь возможность воспроизводить его по мере надобности безо всяких задержек, воспользуйтесь методом `prepareToPlay` для предварительной загрузки ресурсов звука, тем самым подготовив звук для воспроизведения. Можете не вызывать этот метод, он будет вызван автоматически при воспроизведении звука. Однако в этом случае звук может воспроизвестись с небольшой задержкой, которая требуется для подгрузки ресурсов:

```
[ player prepareToPlay ];
```

Когда вы, наконец, будете готовы воспроизводить звук, воспользуйтесь методом проигрывателя `play`:

```
[ player play ];
```

Также вы можете остановить звук в любой момент с помощью метода `stop`:

```
[ player stop ];
```

Методы-делегаты

Протокол `AVAudioPlayerDelegate` описывает методы, которые вы можете назначить делегату, чтобы понимать, когда звук закончит воспроизведение, перехватывать ошибки и получать уведомления о прерываниях.

При вызове метода `play` управление сразу же возвращается вашей программе, пока звук воспроизводится в фоне. Чтобы получить уведомление, когда воспроизведение звука завершится, назначьте свойству проигрывателя `delegate` делегата:

```
player.delegate = self;
```

Когда воспроизведение звука завершится, вызовется метод делегата `audioPlayerDidFinishPlaying`. Чтобы получать уведомления, поместите этот метод в код вашего делегата. В параметрах метода имеется булево значение, определяющее, было ли воспроизведение звука успешным:

```
- (void)audioPlayerDidFinishPlaying:(AVAudioPlayer *)player
  successfully:(BOOL)flag
{
  /* Дополнительный код, выполняющийся после
   * завершения работы проигрывателя */
}
```

Если во время декодирования звука произойдет какая-либо ошибка, то будет вызван метод `audioPlayerDecodeErrorDidOccur`. Вы можете использовать его для изящной обработки ошибок, возникающих во время воспроизведения сэмпла:

```
- (void)audioPlayerDecodeErrorDidOccur:(AVAudioPlayer *)player
  error:(NSError *)error
{
  /* Код обработки ошибок декодера */
}
```

Если работу проигрывателя прерывает входящий звонок, блокировка устройства или какое-либо другое событие, то в начале и в конце прерывающего события уведомляются два метода-делегата. Они могут перезапустить проигрыватель или выполнить какое-либо другое действие, например, попросить вашего пользователя не принимать телефонных звонков, пока вы воспроизводите звук (это нервирует!):

```
- (void)audioPlayerBeginInterruption:(AVAudioPlayer *)player {
  /* Код для обработки прерываний */
}

- (void)audioPlayerEndInterruption:(AVAudioPlayer *)player {
  /* Код для обработки конца прерывания */
}
```

Снятие измерений

Класс `AVAudioPlayer` может передавать измеряемые значения, что позволяет вашему приложению считывать выходные уровни воспроизводимого вами звука. Выходные уровни могут использоваться для отображения визуальных индикаторов звука во время его воспроизведения. Вы можете считывать как средний уровень мощности (в децибелах), так и уровень максимальной (пиковой) мощности. Чтобы разрешить снятие измерений, задайте свойство `meteringEnabled`:

```
player.meteringEnabled = YES;
```

Во время воспроизведения звука может быть вызван метод `updateMeters` для обновления значений измерений:

```
[ player updateMeters ];
```

Затем могут быть считаны средний и пиковый уровни мощности для каждого канала. Значения возвращаются как значения с плавающей точкой, представляя количество децибел для каждого канала. Как правило, эти значения находятся в диапазоне от –100,0 до 0,0:

```
for (int i=0; i<player.numberOfChannels; i++) {
  float power = [ player averagePowerForChannel: i ];
  float peak = [ player peakPowerForChannel: i ];
}
```

Создание волюметра: AVMeter

Этот пример использует преимущества функциональности снятия измерений, предоставляемой библиотекой AVFoundation, для отображения на экране пары волюметров (VU meters) во время воспроизведения звукового содержимого.

Рис. 6.1. Пример AVMeter

Проект по умолчанию воспроизводит файл sample.mp3, а также для отображения волюметров загружает изображения avgMeterImage.png и peakMeterImage.png. Все это входит в состав примеров данной книги, размещенных в Интернете, или же вы можете использовать собственные файлы. Данный пример отображает панель навигации с кнопкой **Play**. После нажатия этой кнопки начинает воспроизводиться сэмпл, а его замеряемые уровни считываются с заданной периодичностью. Измерители на экране (рис. 6.1) обновляются измерениями, считываемыми из `AVAudioPlayer`.

Вы можете скомпилировать это приложение, показанное в листингах 6.1—6.7, с помощью SDK, создав проект приложения AVMeter на базе представления. Помимо этого, вам придется добавить в ваш проект новые файлы: AVMeterView.h и AVMeterView.m. Вы можете сделать это, выбрав в меню **File** команду **New File**, а затем в группе шаблонов **Cocoa Touch Classes**, расположенной под **iPhone OS**, выделить **UIView Subclass**. Если вы хотите увидеть, как создавать все эти объекты с нуля, то удалите код Interface Builder.

Листинг 6.1. Прототипы делегата приложения AVMeter (AVMeterAppDelegate.h)

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


@class AVMeterViewController;


@interface AVMeterAppDelegate : NSObject <UIApplicationDelegate> {
  UIWindow *window;
  UINavigationController *navigationController;
  AVMeterViewController *viewController;
}


@property (nonatomic, retain) IBOutlet UIWindow *window;
@property (nonatomic, retain) IBOutlet AVMeterViewController
*viewController;


@end
```

Листинг 6.2. Делегат приложения AVMeter (AVMeterAppDelegate.m)

```
#import "AVMeterAppDelegate.h"
#import "AVMeterViewController.h"
```

```
@implementation AVMeterAppDelegate

@synthesize window;
@synthesize viewController;

- (void)applicationDidFinishLaunching:(UIApplication *)application {
  CGRect screenBounds = [ [ UIScreen mainScreen ] bounds ];

  self.window = [ [ [ UIWindow alloc ] initWithFrame: screenBounds ]
    autorelease
  ];

  viewController = [ [ AVMeterViewController alloc ] init ];
  navigationController = [ [ UINavigationController alloc ]
    initWithRootViewController: viewController
  ];

  [ window addSubview: [ navigationController view ] ];
  [ window makeKeyAndVisible ];
}

- (void)dealloc {
  [ navigationController release ];
  [ viewController release ];
  [ window release ];
  [ super dealloc ];
}

@end
```

Листинг 6.3. Прототипы контроллера представлений AVMeter (AVMeterViewController.h)

```
#import <UIKit/UIKit.h>
#import <AVFoundation/AVFoundation.h>
#import "AVMeterView.h"
```

```
@interface AVMeterViewController : UIViewController {
  UIBarButtonItem *navButton;
  AVAudioPlayer *player;
  AVMeterView *meterView;
}
- (void)navButtonWasPressed;
- (void)setNavProperties;

@end
```

Листинг 6.4. Контроллер представлений AVMeter (AVMeterViewController.m)

```
#import "AVMeterViewController.h"

@implementation AVMeterViewController

- (id)init {
  self = [ super init ];
  if (self != nil) {
    NSError *err;

    player = [ [ AVAudioPlayer alloc ]
      initWithContentsOfURL: [ NSURL fileURLWithPath: [
        [ NSBundle mainBundle ] pathForResource: @"sample" ofType:@"mp3"
      ] ] error: &err
    ];

    if (err)
      NSLog(@"Failed to initialize AVAudioPlayer: %@\n", err);

    [ self setNavProperties ];
    [ player prepareToPlay ];
  }
  return self;

}
```

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

  CGRect frame = self.view.frame;
  meterView = [ [ AVMeterView alloc ] initWithFrame: frame ];
  meterView.player = player;

  UIImage *peak = [ [ UIImage alloc ] initWithContentsOfFile: [
    [ NSBundle mainBundle ] pathForResource:
      @"peakMeterImage" ofType:@"png"
  ] ];

  UIImage *avg = [ [ UIImage alloc ] initWithContentsOfFile: [
    [ NSBundle mainBundle ] pathForResource:
      @"avgMeterImage" ofType:@"png"
  ] ];

  meterView.meterPeakPowerImage = peak;
  meterView.meterAveragePowerImage = avg;

  self.view = meterView;
}

- (void)setNavProperties {
  navButton = [ [ [ UIBarButtonItem alloc ]
    initWithTitle: (player.playing == YES) ? @"Stop" : @"Play"
    style: UIBarButtonItemStylePlain
     target: self
     action:@selector(navButtonWasPressed)
  ] autorelease ];
  self.navigationItem.rightBarButtonItem = navButton;

  if (player.playing == YES)
    self.title = @"Playing";
  else
    self.title = @"Stopped";
}
```

```
- (void)navButtonWasPressed {
  if (player.playing == YES) {
    [ player stop ];
    [ meterView stopUpdating ];
  } else {
    [ player play ];
    [ meterView startUpdating ];
  }
  [ self setNavProperties ];
}

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

Листинг 6.5. Прототипы представления AVMeter (AVMeterView.h)

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

@interface AVMeterView : UIView {
  AVAudioPlayer *player;
  float cachedAveragePowerForChannel[2], cachedPeakPowerForChannel[2];
  CGRect avgMeterFrame[2], peakMeterFrame[2];
  UIImage *meterAveragePowerImage, *meterPeakPowerImage;
  BOOL updating;

  float meterSpacing;
  float meterHorizontalBorder;
  float meterVerticalBorder;
}
- (id)initWithFrame:(CGRect)frame;
- (void)startUpdating;
- (void)stopUpdating;
```

```
@property(nonatomic,assign) AVAudioPlayer *player;
@property(nonatomic,assign) UIImage *meterAveragePowerImage;
@property(nonatomic,assign) UIImage *meterPeakPowerImage;

@property(nonatomic,assign) float meterSpacing;
@property(nonatomic,assign) float meterHorizontalBorder;
@property(nonatomic,assign) float meterVerticalBorder;

@end
```

Листинг 6.6. Представление AVMeter (AVMeterView.m)

```
#import <CoreGraphics/CoreGraphics.h>
#import "AVMeterView.h"

@implementation AVMeterView
@synthesize player;
@synthesize meterAveragePowerImage;
@synthesize meterPeakPowerImage;
@synthesize meterSpacing;
@synthesize meterHorizontalBorder;
@synthesize meterVerticalBorder;

- (id)initWithFrame:(CGRect)frame {
  self = [ super initWithFrame: frame ];
  if (self != nil) {
    player = nil;
    updating = NO;

    meterAveragePowerImage = nil;
    meterPeakPowerImage = nil;

    meterSpacing = 40.0;
    meterHorizontalBorder = 20.0;
    meterVerticalBorder = 20.0;
```

```
    for(int i = 0; i < 2; i ++) {
      cachedPeakPowerForChannel[i] = 0.0;
      cachedAveragePowerForChannel[i] = 0.0;
    }

    /* Рассчитываем положения измерителя */
    avgMeterFrame[0] = CGRectMake(meterHorizontalBorder,
                                  meterVerticalBorder,
      (frame.size.width / 2) - (meterSpacing/2) - meterHorizontalBorder,
      (frame.size.height) - meterVerticalBorder);
    avgMeterFrame[1] = CGRectMake
      (meterHorizontalBorder + (frame.size.width / 2),
       meterVerticalBorder,
      (frame.size.width / 2) - (meterSpacing/2) - meterHorizontalBorder,
       frame.size.height - meterVerticalBorder);

    peakMeterFrame[0] = CGRectMake(avgMeterFrame[0].origin.x,
                                   avgMeterFrame[0].origin.y,
      avgMeterFrame[0].size.width, meterPeakPowerImage.size.height);
    peakMeterFrame[1] = CGRectMake(avgMeterFrame[1].origin.x,
                                   avgMeterFrame[1].origin.y,
      avgMeterFrame[1].size.width, meterPeakPowerImage.size.height);
  }
  return self;
}

- (void)drawRect:(CGRect)rect {
  float averagePowerForChannel[2], peakPowerForChannel[2];
  BOOL renderedBottom;

  if (player.numberOfChannels < 1) {
    return;
  }

  /* Считываем значения измерителя */
  if (!player || player.meteringEnabled == NO) {
    averagePowerForChannel[0] = averagePowerForChannel[1] = 0.0;
```

```
    peakPowerForChannel[0] = cachedPeakPowerForChannel[0];
    peakPowerForChannel[1] = cachedPeakPowerForChannel[1];
  } else {
    int channels = player.numberOfChannels;
    if (channels > 2)
      channels = 2;
    for(int i = 0; i < channels; i ++)
    {
      float db;
      [ player updateMeters ];
      db = [ player peakPowerForChannel: i ];
      peakPowerForChannel[i] = (50.0 + db) / 50.0;
      db = [ player averagePowerForChannel: i ];
      averagePowerForChannel[i] = (50.0 + db) / 50.0;
    }
    if (channels == 1) {
      peakPowerForChannel[1] = peakPowerForChannel[0];
      averagePowerForChannel[1] = averagePowerForChannel[0];
    }
  }

  /* Либо подпрыгиваем в соответствии с новым уровнем,
   * либо уменьшаем измеритель */
  renderedBottom = YES;
  for (int i = 0; i < 2; i ++) {
    if (averagePowerForChannel[i] > cachedAveragePowerForChannel[i])
    {
      cachedAveragePowerForChannel[i] = averagePowerForChannel[i];
    }
    cachedAveragePowerForChannel[i] -= .02;
    if (cachedAveragePowerForChannel[i] < 0) {
      cachedAveragePowerForChannel[i] = 0;
    }

    if (peakPowerForChannel[i] > cachedPeakPowerForChannel[i]) {
      cachedPeakPowerForChannel[i] = peakPowerForChannel[i];
    }
```

```
    cachedPeakPowerForChannel[i] -= .01;
    if (cachedPeakPowerForChannel[i] < 0.0)
      cachedPeakPowerForChannel[i] = 0.0;

    if ( cachedPeakPowerForChannel[i] != 0.0
      || cachedAveragePowerForChannel[i] != 0.0)
    {
      renderedBottom = NO;
    }

    if (meterAveragePowerImage) {
      [ meterAveragePowerImage drawAsPatternInRect:
      CGRectMake(avgMeterFrame[i].origin.x,
          avgMeterFrame[i].origin.y + (avgMeterFrame[i].size.height
      -(avgMeterFrame[i].size.height * cachedAveragePowerForChannel[i])),
        avgMeterFrame[i].size.width,
        avgMeterFrame[i].size.height - (avgMeterFrame[i].size.height
      -(avgMeterFrame[i].size.height * cachedAveragePowerForChannel[i])))
      ];
    }

    if (meterPeakPowerImage) {
      [ meterPeakPowerImage drawAsPatternInRect:
        CGRectMake(peakMeterFrame[i].origin.x,
        peakMeterFrame[i].origin.y + (avgMeterFrame[i].size.height
        -(avgMeterFrame[i].size.height * cachedPeakPowerForChannel[i])),
        peakMeterFrame[i].size.width,
        meterPeakPowerImage.size.height)
      ];
    }
  }

  if (updating == YES || renderedBottom == NO) {
    [ NSTimer scheduledTimerWithTimeInterval: 0.01
                        target: self
```

```
                    selector: @selector(handleTimer:)
                    userInfo: nil
                     repeats: NO ];
  }
}

- (void)handleTimer:(NSTimer *)timer {
  [ self setNeedsDisplay ];
}

- (void)startUpdating {
  updating = YES;
  player.meteringEnabled = YES;
  [ self setNeedsDisplay ];
}

- (void)stopUpdating {
  updating = NO;
  player.meteringEnabled = NO;
  [ self setNeedsDisplay ];
}

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

Листинг 6.7. Функция `main` для AVMeter (main.m)

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

int main(int argc, char *argv[]) {
  NSAutoreleasePool * pool = [ [ NSAutoreleasePool alloc ] init ];
  int retVal = UIApplicationMain(argc, argv, nil, @"AVMeterAppDelegate");
  [pool release];
  return retVal;
}
```

Как это работает

В примере AVMeter представлен повторно используемый класс `AVMeterView`. Вот как работает этот пример:

1. При запуске приложения уведомляется метод `applicationDidFinishLaunching` класса `AVMeterAppDelegate`. Он создает первоначальное окно, контроллер представлений и контроллер навигации.
2. Контроллер представлений, `AVMeterViewController`, создает свой объект `AVAudioPlayer` и порождает экземпляр нашего собственного класса `AVMeterView`, который назначен активным представлением контроллера.
3. Контроллер инициализирует звуковой проигрыватель со звуком сэмпла и прикрепляет его к нашему собственному объекту `AVMeterView` путем задания его свойства `player`. Кроме того, он загружает и задает изображения для графического представления среднего и пикового уровней.
4. Когда пользователь нажимает кнопку **Play**, внутри контроллера представлений вызывается метод `navButtonWasPressed`. Тем самым звуковому проигрывателю предписывается начать воспроизведение, а также вызывается метод `startUpdating` класса `AVMeterView`.
5. Метод `drawRect` класса `AVMeterView` подменяется и вызывается всякий раз, когда требуется обновить экран. Метод `drawRect` считывает значения измерений проигрывателя и подсчитывает положение столбцов измерителя. Затем он вызывает таймер, созданный для осуществления обновлений экрана.

Для дальнейшего изучения

Новый класс AVFoundation обладает несколькими привлекательными характеристиками. В заголовочных файлах вашего SDK проверьте наличие его прототипов. Вы найдете их в папке /Developer/Platforms/iPhoneOS.platform внутри каталога Headers библиотеки AVFoundation.

Аудиосервисы

Для простых немикшированных звуков библиотека Audio Toolbox предоставляет аудиосервис (audio service) в стиле C. Вы можете воспроизводить простые звуки с помощью функции `AudioServicesPlaySystemSound`. При воспроизведении простых звуков должны соблюдаться несколько правил.

Длительность звучания не должна превышать 30 секунд, а звуки должны быть представлены в формате PCM или IMA4. Кроме того, звуковой файл должен храниться в файле caf, aif или wav. Простые звуки не могут воспроизводиться из памяти, а только с диска. Если вы в своей работе над проектом используете iPhone SDK версии не ниже 2.2, то вместо аудиосервисов вам лучше воспользоваться библиотекой AVFoundation.

Помимо приведенных выше ограничений, накладываемых на простые звуки, вы будете иметь весьма ограниченный контроль над способом воспроизведения звуков. Воспроизведение звука будет начинаться незамедлительно и с тем уровнем громкости, который установил пользователь телефона. Вы не сможете зацикливать звуки или управлять стерео. Однако вы сможете вызвать функцию обратного вызова, когда воспроизведение звука будет завершено, таким образом вы сможете освободить ваши звуковые объекты и уведомить ваше приложение.

В листинге 6.8 продемонстрирован аудиосервис, который вызывает на воспроизведение файл sample.wav.

Листинг 6.8. Пример аудиосервиса (SOundExample.m)

```
#include <AudioToolbox/AudioToolbox.h>
#include <CoreFoundation/CoreFoundation.h>

// Эта функция вызывается, когда воспроизведение звука будет завершено
static void SoundFinished (SystemSoundID soundID, void *sample)
{
  /* Я завершил воспроизведение, поэтому освободи ресурсы */
  AudioServicesDisposeSystemSoundID(sample);
  CFRelease(sample);
  CFRunLoopStop(CFRunLoopGetCurrent());
}

// Основной цикл
int main()
{
  /* System Sound ID, под которым мы хотим
   * воспроизводить наш звук */
  SystemSoundID soundID;
  NSURL *sample = [ [ NSURL alloc ] initWithString: @"sample.wav" ];
```

```
  OSStatus err = AudioServicesCreateSystemSoundID (sample, &soundID);
  if (err) {
    NSLog(@"Error occurred assigning system sound!");
    return(-1);
  }

  /* Добавляем функцию обратного вызова на завершение воспроизведения */
  AudioServicesAddSystemSoundCompletion(soundID, NULL, NULL,
    SoundFinished, sample);

  /* Воспроизвели его! */
  AudioServicesPlaySystemSound(soundID);
  CFRunLoopRun();
  return 0;
}
```

Как это работает

Пример аудиосервиса работает следующим образом:

1. При запуске приложения вызывается его функция `main`.
2. Создается объект `NSURL`, указывающий на путь к звуковому файлу.
3. Функция `AudioServicesCreateSystemSoundID` вызывается для создания из звукового файла системного звука.
4. Функция обратного вызова на завершение воспроизведения, `SoundFinished`, регистрируется как обратный вызов.
5. Идентификатору системного звука, назначенному нашему звуку, указывается воспроизвести звук путем вызова `AudioervicesPlaySystemSound`.
6. Когда воспроизведение звука завершится, будет вызвана функция `SoundFinished`, которая разрушит все звуковые объекты.

Аудиоочереди

Помимо простых звуков, Audio Toolbox содержит функции низкого уровня для обработки звука на уровне потока битов. Данная библиотека включает

множество API, предоставляющих доступ к необработанным данным внутри звуковых файлов, а также целый ряд средств конвертирования. Если вы пишете игры или другие приложения, активно работающие со звуком, то вам, возможно, потребуется аудиоочередь (audio queue) для формирования выходного цифрового потока или управления микшированием стереоканалов. Такая очередь, в отличие от аудиосервисов, работает с потоками необработанных аудиоданных, а не с самими 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.

Очередь Audio Toolbox является строго "первым прибыл — первым обслужен" (first in, first out — FIFO), т. е. ленточный конвейер воспроизводит сэмплы 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 сэмпла и т. д.).
2. К очереди прикрепляются звуковые буферы, которые будут содержать сами звуковые фреймы для воспроизведения. Звуковой фрейм можно рассматривать как отдельный ящик, заполненный звуком, а сэмпл — 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 частей, вовлеченных в подготовку аудиоочереди и предназначенных для того, чтобы сделать наши примеры более понятными. Все эти различные используемые перемен-

ные 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 аудиобиблиотеки:

- `AudioQueueRef queue` — указатель на объект аудиоочереди, который создаст ваша программа;
- `Uint32 frameCount` — общее число сэмплов, копируемых за одну итерацию аудиосинхронизации. 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 описание аудиопотока:

```
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 сэмпл) стереозвука (два канала) с частотой сэмпла 44 кГц (44 100 Гц).

Наш выходной сэмпл будет представлен в форме двух двухбайтных *коротких* целых чисел (short integers), а, следовательно, четырех полных байтов на фрейм (по два байта для левого и правого каналов).

Частота сэмпла и размер фрейма определяют то, как часто iPhone будет запрашивать звук. С частотой в 44 100 сэмплов 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 аудиоочередь поддерживает все форматы звука, которые поддерживает iPhone. Это следующие 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 аудиоочереди 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 станет вызываться, когда буфер аудиоочереди будет пуст 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 как информацию об аудиоочереди, так и любую другую относящуюся к приложению информацию о воспроизводимых сэмплах;
- `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 режимы;
- `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,
                    AQBufferCallback,
                    &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 та 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. Возвращаясь 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 больше у вас ящиков, тем больше звука вы можете заранее поместить в очередь, чтобы избежать перерывов в воспроизведении (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 подпрыгивает, но пользователь не слышит этого до тех пор, пока он не приземлится.

Когда звук готов к началу воспроизведения, ваш код создает звуковые буферы и заполняет их 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 интенсивной загрузке процессора, более подходящим является использование большего количества буферов для предотвращения перегрузок. Чтобы подготовить буферы с первыми фреймами звуковых данных, каждый буфер предоставляется вашей функции обратного вызова, которая будет заполнять их звуком, только единожды. Это означает, 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. После того как 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 функция обратного вызова (callback function) вашего приложения 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 вашем вызове `AudioQueueNewOutput`. Данная функция обратного вызова — это место, где приложение проделывает свою работу; она заполняет ящик, который переносит ваш выводимый звук к динамикам. Аудиоочередь вызывает эту 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 сэмплов:

```
static void AQBufferCallback(
  void *aqc,
  AudioQueueRef inQ,
  AudioQueueBufferRef outQB)
{
```

Структура обратного вызова, которую вы создали в самом начале, `aqc`, передается как заданный 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 осуществлен доступ. Данный код захватывает указатель на необработанные аудиоданные внутри буфера, в результате чего приложение может писать в него свой звук:

```
short *CoreAudioBuffer = (short *) outOB->mAudioData;
```

Переменная `CoreAudioBuffer` представляет пространство внутри звукового буфера, в который будут копироваться необработанные сэмплы вашего приложения во время каждой синхронизации. Вашему приложению потребуется поддерживать тип "записывающей иглы", чтобы отслеживать, байты какого звука уже были отправлены в аудиоочередь:

```
if (inData->frameCount > 0) {
```

Переменная `frameCount` определяет 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.

Уровень громкости

Сэмплы воспроизводятся посредством аудиоочереди с системным уровнем громкости (volume control), но вы можете настроить величину вашего звукового вывода. В функции обратного вызова, использованной в предыдущем разделе, мы копировали звуковые фреймы из звукового вывода приложения в звуковые буферы при каждой синхронизации. Корректируя эти значения с помощью множителя уровня громкости, вы можете эффективно увеличивать и уменьшать выходной уровень громкости ваших сэмплов:

```
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 * volumeMultiplier;
  coreAudioBuffer[i+1] = sample * volumeMultiplier;
  aqc->playPtr++;
}
```

Когда учитывается громкость, величина сэмпла (уровень исходного сигнала в сэмпле) умножается на значение уровня громкости и таким образом уменьшается или увеличивается в соответствии с уровнем громкости. Если вы хотите, чтобы максимальный уровень был громче, задайте в качестве множителя уровня значение, большее 1,0. Чтобы уменьшить уровень, задайте в качестве множителя десятичное число, меньшее 1,0. Будьте осторожны и не перегрузите ваш аудиовыход, что может привести к искажению звука.

Пример: проигрыватель PCM

В данном примере применяется добрый старый 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 сам принцип. Большинству приложений должна подойти подобная архитектура.

Поскольку необработанный файл PCM не содержит какую-либо информацию об его частоте или размере фрейма, то данный пример будет вынужден сде-

лать собственные предположения на этот счет. Мы будем использовать формат для 16-битных 44-килогерцовых несжатых моноданных PCM. Это задается тремя описаниями, сделанными в самом начале программы:

```
#define BYTES_PER_SAMPLE 2
```

16 бит = 2 байта:

```
#define SAMPLE_RATE 44100
```

44 100 сэмплов в секунду = 44 кГц:

```
typedef unsigned short sampleFrame;
```

`unsigned short` равен двум байтам (на сэмпл).

Если вы не можете найти необработанный файл PCM для выполнения этого примера, то используйте 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 исходит из того, что предоставляемые вами данные являются необработанными, т. е. такими, какие предоставляют игры или приложения другого типа. Заголовки wave-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. Поскольку Leopard также содержит библиотеку Audio Toolbox, то вы можете скомпилировать этот пример как на настольной станции, так и для iPhone:

```
$ gcc -o playpcm playpcm.c \
  -framework AudioToolbox -framework CoreAudio -framework CoreFoundation
```

В листинге 6.9 приведен сам код.

Листинг 6.9. Пример использования 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 определяемую пользователем структуру AQCallbackStruct, чья конструкция декларируется в начале программы. Эта структура содержит указатели на аудиоочередь, звуковые буферы и память, хранящую содержимое загруженного файла. Кроме того, она содержит длину сэмпла и целое число playPtr, которое выступает в роли записывающей иглы, определяя последний сэмпл, скопированный в звуковой буфер.
4. Инициализируется и запускается новая звуковая очередь. Чтобы синхронизировать первые сэмплы в памяти, вызывается функция обратного вызова по одному разу для каждого используемого звукового буфера. Затем стартует аудиоочередь. После этого программа ждет, когда закончится воспроизведение текущего сэмпла.
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 в звуковой буфер следующие звуковые фреймы из памяти для воспроизведения. Поскольку необработанные дорожки PCM являются моно, то одинаковые данные копируются как в левый, так и в правый выходной каналы.
7. Когда playPtr превысит длину звукового сэмпла, то цикл ожидания в playpcm прервется, в результате чего функция вернется обратно к main для освобождения ресурсов и выхода.

Для дальнейшего изучения

- 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 звуковой сэмпл теперь имеет длину в один байт, а канал аудиоочереди — 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 практически точно так же, как его воспроизводит аудиоочередь, однако очередь создается как очередь записи, предоставляя приложению выходные данные, а не принимая входные данные с динамиков. 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 аудиоочererди, 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 библиотеки о том, какой формат и частоту сэмпла вы бы хотели, но теперь вы отвечаете не за заполнение ящиков, а за их опустошение и запись на диск или какое-либо другое устройство хранения. Для записи непосредственно в файл, а не копирования его в память, вы будете использовать в этом примере функции `AudioFile` из Audio Toolbox. Очередь записи остается строго "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 сэмпла 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 сэмпл — это отдельная часть цифрового звука внутри ящика.
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 (или другое устройство), а затем отправку ящика по кругу для следующего заполнения.

Структура аудиоочереди

Как мы уже объясняли ранее, платформа Audio Toolbox использует вызовы низкоуровневых функций 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. Рассматривайте ее как контекст. Структура `AQCallbackStruct`, приведенная далее, похожа на версию обратного вызова данной структуры, но с несколькими добавленными фрагментами:

```
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 аудиобиблиотеки:

- `AudioStreamBasicDescription mDataFormat` — информация о формате аудио, которое будет записываться;
- `AudioQueueRef queue` — указатель на объект аудиоочереди, который создаст ваша программа;
- `AudioQueueBufferRef mBuffers` — массив, содержащий общее количество звуковых буферов, которые будут использоваться;
- `AudioFileID outputFile` — указатель на выходной файл, который будет записан после того, как запишется звук;
- `unsigned long frameSize` — общее количество сэмплов для копирования за одну аудиосинхронизацию. Это полностью отдается на откуп реализующему;
- `long long recPtr` — числовой указатель на текущую позицию записывающей "иголки" на основе того, какие звуковые данные 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 ли отправлять ящики назад для повторного заполнения. Когда придет время завершить запись, это значение должно быть установлено вашим приложением в ноль.

Прежде чем создать аудиоочередь, необходимо инициализировать описание аудиовхода, который должно получать ваше приложение:

```
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 записи 16-битного (по два байта на сэмпл) стереозвука (два канала) с частотой сэмпла 44 кГц (44 100). Выходной сэмпл будет представлен в виде двух двухбайтных *коротких* (short) целых чисел, следовательно, по четыре байта на фрейм (по два байта на левый и правый каналы).

Частота сэмпла и размер фрейма определяют то, как часто ваше приложение будет получать звук. При частоте 44 100 сэмплов в секунду приложение можно заставить синхронизировать звук каждую 1/60-ую часть секунды, задав размер фрейма в 735 сэмплов (44 100/60 = 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 использовался формат PCM (необработанные данные), но аудиоочередь допукает и многие другие аудиоформаты, поддерживаемые iPhone.

Такими форматами являются:

- `kAudioFormatLinearPCM`;
- `kAudioFormatAppleIMA4`;
- `kAudioFormatMPEG4AAC`;
- `kAudioFormatULaw`;
- `kAudioFormatALaw`;
- `kAudioFormatMPEGLayer3`;
- `kAudioFormatAppleLossless`;
- `kAudioFormatAMR`.

Подготовка аудиоввода

После того как свойства аудиоочереди определены, можно подготовить объект новой аудиоочереди. За подготовку входного (записывающего) канала и прикрепление его к очереди отвечает функция `AudioQueueNewInput`. Прототип функции выглядит следующим образом:

```
AudioQueueNewInput(
  const AudioStreamBasicDescription *inFormat,
  AudioQueueInputCallback inCallbackProc,
  void * inUserData,
  CFRunLoopRef inCallbackRunLoop,
  CFStringRef inCallbackRunLoopMode,
  UInt32 inFlags,
  AudioQueueRef * outAQ);
```

Здесь:

- `inFormat` — указатель на структуру, описывающую аудиоформат, который будет записываться. Вы определили эту структуру ранее как член типа данных `AudioStreamBasicDescription` в рамках структуры `AQCallbackStruct`;
- `inCallbackProc` — имя функции обратного вызова, которая будет вызываться, когда буфер аудиоочереди окажется заполненным записанными данными. Функция обратного вызова отвечает за выполнение таких действий со звуковым буфером, как запись его на диск, а затем отправку буфера обратно по кругу за новыми данными;

- inUserData — указатель на данные, которые разработчик при желании может передавать в функцию обратного вызова. Наш пример 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 сэмплах;
- `inCallbackRunLoopMode` — сообщает аудиоочереди о том, какое должно быть зацикливание аудио. Значение `NULL` указывает на то, что функция обратного вызова должна вызываться каждый раз, когда звуковой буфер заполнен. Для запуска обратного вызова при других условиях существуют дополнительные режимы;
- `inFlags` — не используется, зарезервировано;
- `outAO` — когда функция `AudioQueueNewOutput` возвращает рзультат, 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 представлен далее. В этом примере название нашей функции обратного вызова — `AQInputCallback`. Это та функция, которая будет отвечать за прием записываемого звука, переданного в ваше приложение, и запись его на диск:

```
AudioQueueNewInput(&aqc.mDataFormat,
                   AQInputCallback,
                   &aqc,
                   NULL,
                   kCFRunLoopCommonModes,
                   0,
                   &aqc.queue);
```

Звуковые буферы

Звуковой буфер (sound buffer) содержит звуковые данные с микрофона, которые находятся по пути к вашему приложению (выводящему устройству). Возвращаясь к нашей концепции ящика на ленточном конвейере, 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 имеется у вас 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 микрофона перехват звука, направляющегося в приложение, занимает больше времени. Это может стать проблемой, если вы пишете синтезатор речи или приложение другого типа, требующего максимального приближения к звуку реального времени.

Когда запись готова начаться, создаются звуковые буферы и помещаются в аудиоочередь. 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 интенсивной загрузке процессора, более подходящим является использование большего количества буферов для предотвращения перегрузок:

```
#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 — это пример записи звука, а не его воспроизведения, то буфер должен быть сначала поставлен в очередь (отправлен обратно по ленточному конвейеру), чтобы аудиобиблиотека могла заполнить его записанными данными. После заполнения библиотека автоматически вызовет вашу функцию обратного вызова.

Теперь очередь готова к работе, что приводит к включению ленточного конвейера, отправляющего звуковые буферы от микрофона к вам. Как только это произойдет, функция обратного вызова освобождает буферы от их содержимого (нет, память не обнуляется), а ящики возвращает по ленточному конвейеру на повторное заполнение:

```
AudioQueueStart(aqc.queue, NULL);
```

Далее, когда вы готовы отключить запись звука, деактивируйте звуковую очередь с помощью функций `AudioQueueStop` и `AudioQueueDispose`. Функция `AudioQueueStop` только останавливает очередь, оставляя ее в таком состоянии, в котором она позднее может быть перезапущена.

Но если аудиоочередь удаляется, то она удаляется из памяти и не может быть перезапущена:

```
AudioQueueStop(aqc.queue, true);
AudioQueueDispose(aqc.queue, true);
```

Функция обратного вызова

Теперь аудиоочередь запущена, и вашему приложению периодически предоставляется звуковой буфер, содержащий данные. До сих пор еще не было объяснено, как это происходит. После того как буфер будет заполнен записываемыми данными, аудиоочередь вызывает функцию обратного вызова (callback function), заданную вами в качестве второго аргумента в `AudioQueueNewInput`. Данная функция обратного вызова — это место, где приложение проделывает свою работу; она опустошает ящик, который переносит вывод микрофона, и помещает его обратно в очередь. При вызове ваша функция обратного вызова опустошает буфер аудиоочереди, копируя самые последние звуковые фреймы в соответствующие им места назначения, в нашем примере — в файл:

```
static void AQInputCallback (
  void *aqr,
  AudioQueueRef inQ,
  AudioQueueBufferRef inQB,
  const AudioTimeStamp *timestamp,
  unsigned long frameSize,
  const AudioStreamPacketDescription *mDataFormat)
{
```

Структура обратного вызова, которую вы создали в самом начале, `aqc`, передается в вашу функцию обратного вызова как определяемый пользователем аргумент, вместе с указателями на саму аудиоочередь и буфер аудиоочереди для опустошения:

```
AQCallbackStruct *aqc = (AQCallbackStruct *)aqr;
```

Поскольку структура `AQCallbackStruct` считается пользовательскими данными, то аудиоочередь представляет ее функции обратного вызова как указатель на объект неизвестного типа (void pointer). Она должна быть приведена к структуре `AQCallbackStruct` (здесь называемой `aqc`) до того, как к ней может быть осуществлен доступ.

Осуществление доступа к необработанным данным

Чаще всего вы будете писать аудио напрямую в файл, но если вы собираетесь осуществить доступ к необработанным данным (raw data) внутри буфера, то можете обратиться к буферу необработанных входных данных:

```
short *CoreAudioBuffer = (short *) inQB->mAudioData;
```

Переменная `CoreAudioBuffer` представляет пространство внутри звукового буфера, в который будут копироваться необработанные сэмплы с микрофона во время каждой синхронизации. Вашему приложению потребуется поддерживать тип "записывающей иглы", чтобы отслеживать то, байты какого звука уже были отправлены в аудиоочередь. Вот пример копирования данных в выделенную память:

```
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);
}
```

Запись в файл

Для записи в файл вы будете использовать множество функции `AudioFile` из Audio Toolbox. Чтобы подготовить аудиофайл, сначала вам потребуется описать формат файла. Приведенный далее код определяет свойство, необходимое для аудиофайла формата AIFF:

```
AudioFileTypeID fileFormat = kAudioFileAIFFType;
```

Используйте структуру `CFURL` для хранения действительного пути к аудиофайлу:

```
CFURLRef filename = CFURLCreateFromFileSystemRepresentation(
        NULL,
       (const unsigned char *) path_to_file,
        strlen (path_to_file),
        false);
```

> ***Внимание!***
>
> С помощью функции `NSHomeDirectory` или аналогичной убедитесь в том, что выбранный вами путь к файлу существует в рамках песочницы вашего приложения. Записывать звуковой файл вне вашей песочницы запрещено.

Наконец, сам аудиофайл вы создаете с помощью вызова `AudioFileCreateWithURL`, содержащего имя файла и только что созданные свойства формата. Указатель на файл записан в структуру `AQCallbackStruct`, поэтому, когда появится звук для записи, вы будете знать, как осуществить доступ к файлу:

```
AudioFileCreateWithURL(filename,
                        fileFormat,
                       &aqc.mDataFormat,
                        kAudioFileFlags_EraseFile,
                       &aqc.mAudioFile);
```

По мере записи новых аудиосэмплов вы будете осуществлять запись в этот файл с помощью функции `AudioFileWritePackets` — еще одной встроенной в Audio Toolbox функции, специально предназначенной для записи аудиопакетов в файл. Как все это работает, вы увидите в следующем примере.

Пример: магнитофон

Продолжая традицию в духе доброго старого C, данный пример может запускаться из командной строки с указанием имени файла и длительности как на iPhone, так и на настольной системе. Он записывает данные с микрофона в течение указанной длительности и сохраняет их в файл с указанным именем.

Поскольку Leopard также содержит библиотеку Audio Toolbox, то данный пример может быть скомпилирован как для iPhone, так и для настольной системы. В листинге 6.10 приведен соответствующий код:

```
$ gcc -o recorder recorder.c -framework AudioToolbox -framework
CoreFoundation
```

Листинг 6.10. Пример магнитофона (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 звуковые буферы
     не помещаем в очередь повторно */
  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 = kAudioFileAIFFType;
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 и длительность из списка аргументов (передаваемые в командной строке).
2. Функция `main` создает нашу задаваемую пользователем структуру `AQCallbackStruct`, чья конструкция декларируется в самом начале программы. Эта структура хранит указатели на очередь записи, звуковые буферы и созданный выходной файл. Кроме того, она содержит длину сэмпла и целое число `recPtr`, которое выступает в роли записывающей иглы, определяя последний сэмпл, записанный на диск.
3. Инициализируется и запускается новая очередь записи. Инициализируется и ставится в эту очередь каждый звуковой буфер. Затем стартует аудиоочередь. После этого программа ждет, когда закончится запись сэмпла.
4. По мере записи аудио звуковые буферы отправляются к функции обратного вызова, где они один за другим заполняются. Каждый раз, когда какой-либо буфер готов к заполнению, вызывается функция `AQInputCallback`.

5. Функция `AQInputCallback` увеличивает `recPtr` и копирует звуковой фрейм на диск.

Для дальнейшего изучения

- Модифицируйте данный пример для синхронизации через интервалы в одну секунду.
- Проверьте наличие AudioFile.h в Mac OS X Leopard на настольной системе. Вы сможете найти его в папке: /System/Library/Frameworks/AudioToolbox.framework/Headers/.

Вибрация

Вам может быть интересно, как создать некоторую суету и суматоху без воспроизведения какого-либо звука, т. е. как заставить iPhone вибрировать. Если владелец iPhone положил его на стол в комнате переговоров, то вибрация может породить различные раздражающие звуки, привлекая внимание пользователя. Для этого вы можете использовать имеющийся в аудиосервисе, описанном в начале этой главы, идентификатор. В листинге 6.11 показано, как это делается.

Листинг 6.11. Пример кода для создания вибрации

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

- (void)vibrate
{
  AudioServicesPlaySystemSound(kSystemSoundID_Vibrate);
}
```

Глава 7

Сетевое программирование с CFNetwok

Одной из самых сильных сторон iPhone является его способность предоставлять расширенные функциональные возможности по постоянно действующему подключению к Интернету. Хотя iPhone поддерживает стандартный набор функций C для сетевого программирования, Apple предоставила еще и библиотеку CFNetwork, отвечающую за такую функциональность, как разрешение имен (name resolution), поддержка сокетов (socket connectivity) и базовые взаимодействия протоколов.

Вы можете рассматривать использование CFNetwork вместо стандартных сокетов BSD, если вашему приложению необходим цикл обработки сообщений, т. е. вы сможете использовать CFNetwork без запуска параллельных потоков или написания собственных опрашивающих процедур. Кроме того, библиотека CFNetwork добавляет простой механизм для обработки считывания и записи посредством потоков сокетов и поддерживает общепринятые протоколы, такие как HTTP и FTP. Это позволяет вам сфокусировать свое внимание на более важных аспектах вашего кода, освобождая вас от необходимости поддерживать отдельные протоколы. Поскольку CFNetwork поддерживается и в Leopard, то вы можете легко переносить ваш код для работы в сетях на настольные системы.

Чтобы воспользоваться библиотекой CFNetwork, вам потребуется добавить ее в ваш проект Xcode. Для этого щелкните правой кнопкой мыши по папке Frameworks в вашем проекте, а затем в появившемся контекстном меню вы-

берите команду **Add | Framework**. Перейдите к папке CFNetwork.Framework, а затем нажмите кнопку **Add**.

Программирование сокетов

Чаще всего библиотека CFNetwork используется для организации взаимодействия посредством сетевых сокетов. Сокет (socket) — это канал взаимодействия между двумя конечными точками; рассматривайте его как нить, соединяющую две жестяные банки. Сокеты могут быть установлены между компьютерами в рамках сети (например, как HTTP-подключение между iPhone и Web-сервером) или же локально между приложениями на одном и том же устройстве (например, как между двумя программами, использующими совместно данные). Конечную точку, отвечающую за инициирование подключения, как правило, называют *клиентом* (client), а конечную точку, получающую и обслуживающую подключение, называют *сервером* (server). Еще одна сторона данной парадигмы — это одноранговое сетевое взаимодействие (Peer-to-peer networking). Одноранговые сети (peer-to-peer networks) для совместного использования ресурсов применяют специально созданные подключения между автономными клиентскими машинами, а не подключаются к центральному серверу для получения необходимых ресурсов. Однако во многих ситуациях центральный сервер все-таки используется для координации узлов сети.

Типы сокетов

Существуют два основных типа сокетов: UDP и TCP.

Сокет UDP (User Datagram Protocol) служит для отправки получателю коротких сообщений, называемых *дейтаграммами* (datagrams). Дейтаграммы являются отдельными пакетами данных, отправляемыми и принимаемыми без каких-либо "уведомлений о доставке". При этом не дается никакой гарантии в том, что получатель действительно получит какой-либо конкретный пакет, а несколько пакетов будут получены в определенном порядке. Дейтаграммы, как правило, считаются ненадежными, как, например, ненадежным является почтовый голубь. Такой тип взаимодействия применяется при отправке коротких запросов и/или требующих ответа сообщений, которые не нуждаются в аутентификации, например, DNS (разрешение имен), а также некоторыми протоколами, в которых потеря пакетов не критична, например, потоки "жи-

вого" видео (в реальном времени) или многопользовательские игры в Интернете, когда какое-либо прерывание может быть проигнорировано.

Другой тип сокетов, TCP (Transmission Control Protocol), является гораздо более распространенным, поскольку он предоставляет среду для полноценного, структурированного "разговора" между двумя конечными точками. TCP-подключения предоставляют средства, позволяющие получить гарантию в том, что сообщения были получены, а пакеты были доставлены в правильном порядке. TCP используется протоколами HTTP, FTP и другими, в которых данные должны надежно отправляться и приниматься в соответствующем порядке. Чтобы отслеживать расположение пакетов, TCP использует порядковые номера, указывающие на порядок следования каждого пакета. Это не только поддерживает порядок в вашем разговоре, но и разумный уровень защиты против получения доступа путем обмана (spoofing), т. е. подделки пакетов злоумышленником.

Объект *CFSocket*

Объект `CFSocket` является базовой структурой, использованной для создания UDP- и TCP-сокетов, и применяется для установки, осуществления взаимодействия и разрыва соединения. Поскольку библиотека CFNetwork основана на C, то `CFSocket` скорее структура, нежели класс, и все вызовы ее функций в качестве первого аргумента содержат сокет.

Создание новых сокетов

Для создания UDP- и TCP-сокетов можно использовать функцию `CFSocketCreate`. Она вызывается с указанием всей информации, необходимой для создания сокета, включая тип распределителя памяти, семейство протоколов (IPv4 или IPv6) и тип сокета. Сокет может работать в фоновом режиме, если добавить его обработку в главный цикл (цикл обработки сообщений) приложения, поэтому вы сможете предоставить функцию обратного вызова, которая будет вызываться во время определенных событий сокета. Это позволит вам сфокусировать свои усилия на самом приложении, а не на написании вашего собственного цикла для ожидания подключений и данных. Вот прототип для `CFSocketCreate`:

```
CFSocketRef CFSocketCreate (
    CFAllocatorRef allocator,
    SInt32 protocolFamily,
```

```
    SInt32 socketType,
    SInt32 protocol,
    CFOptionFlags callBackTypes,
    CFSocketCallBack callout,
    const CFSocketContext *context
);
```

Здесь:

- `CFAllocatorRef allocator` задает тип распределителя памяти для создания нового сокета. Чтобы задать распределитель по умолчанию, используйте значение `NULL` или `kCFAllocatorDefault`. Другими типами распределителей являются:
  - `kCFAllocatorSystemDefault` — системный распределитель по умолчанию, который Apple строго не рекомендует использовать;
  - `kCFAllocatorMalloc`, который использует `malloc()`, `realloc()` и `free()`;
  - `kCFAllocatorMallocZone`, который выделяет пространство в несканируемой части памяти;
  - `kCFAllocatorNull`, который не выделяет и не освобождает память, а используется в случаях, когда данные не должны перемещаться.

  Практически всегда вы будете использовать распределитель по умолчанию за исключением весьма специфических ситуаций;
- `SInt32 protocolFamily` — семейство протоколов для сокета. По умолчанию используется `PF_INET` (Internet Protocol, известный также как IPv4). Для сокетов IPv6 используйте `PF_INET6`;
- `SInt32 socketType` определяет тип сокета. Для сокетов TCP используйте `SOCK_STREAM`, а для сокетов UDP — `SOCK_DGRAM`;
- `SInt32 protocol` — протокол, который будет использоваться для сокета. Это может быть либо `IPPROTO_TCP`, либо `IPPROTO_UDP` и должен соответствовать заданному типу сокета;
- `CFOptionFlags callBackTypes` — главный цикл обработки может вызывать функции обратного вызова для событий сокета различных типов. Например, функция обратного вызова может быть вызвана при подключении сокета или при наличии данных. Аргумент `callBackTypes` создается с помощью битовой маски, которую вы можете задать с помощью по-

битовой операции `OR`. Прототип Apple определяет следующий список флагов:

```
enum CFSocketCallBackType {
    kCFSocketNoCallBack = 0,
    kCFSocketReadCallBack = 1,
    kCFSocketAcceptCallBack = 2,
    kCFSocketDataCallBack = 3,
    kCFSocketConnectCallBack = 4,
    kCFSocketWriteCallBack = 8
};
typedef enum CFSocketCallBackType CFSocketCallBackType;
```

- `CFSocketCallBack callout` определяет функцию, которая должна вызываться, когда начинается одно из указанных в `callBackTypes` событий. Вместо написания вашего собственного цикла обработки событий установления соединения, отправки и получения данных использование главного цикла будет вызывать эту функцию всякий раз, когда будет происходить одно из желаемых событий;
- `const CFSocketContext *context` — специальная структура, содержащая контекст `CFSocket`, который может инкапсулировать указатель на ваши собственные задаваемые пользователем данные для сокета. Об этом вы узнаете в следующем разделе;
- `CFTimeInterval timeout` определяет время ожидания подключения. Если задать отрицательное значение, то подключение будет устанавливаться в фоновом режиме, позволяя вашей программе продолжать свою работу. В этом случае функция обратного вызова будет вызываться, когда подключение будет создано.

Создание сокетов из существующего сокета

Объект `CFSocket` можно создать из существующего встроенного (native) сокета с помощью функции `CFSocketCreateWithNative`. Эта функция очень похожа на функцию `CFSocketCreate`, но принимает в качестве одного из своих аргументов существующий сокет. Это может оказаться полезным в случаях, когда у вас для построения сокета имеется оставшийся вам по наследству код, а вы хотите встроить его в библиотеку CFNetwork. Вот прототип этой функции:

```
CFSocketCreateWithNative (
    CFAllocatorRef allocator,
```

```
    CFSocketNativeHandle sock,
    CFOptionFlags callBackTypes,
    CFSocketCallBack callout,
    const CFSocketContext *context
);
```

В приведенном выше примере сразу можно заметить, что аргумент встроенного сокета описывается не как `int`, что является стандартом для большинства систем, а как `CFSocketNativeHandle`. Тип данных `CFSocketNativeHandle` определен как встроенный тип данных операционной системы для сокетов на базе C, поэтому он, как правило, в большинстве систем будет разрешаться как `int`.

Функции сокетов

После создания сокета над ним можно выполнить целый ряд функций. Вы можете использовать функции, характерные для библиотеки CFNetwork, и специальную функцию `CFSocketGetNative`. Кроме того, вы можете работать и на низкоуровневом встроенном сокете данного сокета для выполнения встроенных C-функций сокета. Для объектов `CFSocket` существуют следующие функции.

- `CFSocketGetNative` возвращает встроенный сокет системы, над которым вы можете выполнить встроенное множество сокетных C-операций. Как правило, эта функция в большинстве систем имеет тип данных `int`. В приведенном далее примере вы увидите вызов встроенной функции `setsockopt()`. Это позволяет вам встраиваться в библиотеку CFNetwork, не жертвуя никакой частью функциональности, но при этом сохраняя совместимость с унаследованным кодом.
- `CFSocketConnectToAddress` вызывает запрос на подключение на локальном сокете. Используется для подключения сокета к ожидающему подключения (серверному) сокету, например, к Web-серверу.
- `CFSocketCopyAddress` возвращает локальный адрес `CFSocket`. Это удобно при определении того, с какого IP-адреса или адресов ожидает подключения ваш сокет.
- `CFSocketCopyPeerAddress` возвращает адрес удаленного сокета, к которому подключен `CFSocket`. Она предоставляет IP-адрес удаленной конечной точки подключения для событий, в которых ваше приложение выступает в качестве сервера.

- `CFSocketCreateRunLoopSource` создает исходный объект цикла обработки сообщений для объекта `CFSocket`. Вы увидите, как это работает, в следующем примере.

Разрешение/запрет обратных вызовов

Обратные вызовы для объектов `CFSocket` могут быть разрешены или запрещены на уровне программиста. Это полезно в случаях, когда обратный вызов для сокета ведет себя по-разному в зависимости от состояния сокета. Это можно сделать с помощью функций `CFSocketDisableCallBacks` и `CFSocketEnableCallBacks`.

Обе функции в качестве аргументов принимают сокет и набор уже изученных вами флагов функции обратного вызова, скомбинированных посредством побитовой операции `OR`. Чтобы запретить прием и считывание обратных вызовов для заданного сокета, ваш код должен выглядеть примерно так:

```
CFSocketDisableCallBacks(mySocket,
    kCFSocketAcceptCallBack | kCFSocketReadCallback);
```

Чтобы потом снова разрешить обратные вызовы, просто замените название функции на `CFSocketEnableCallBacks`:

```
CFSocketEnableCallBacks(mySocket,
    kCFSocketAcceptCallBack | kCFSocketReadCallback);
```

Отправка данных

Библиотека CFNetwork предоставляет абстрактную процедуру для отправки данных, которая помогает упростить этот процесс. Чтобы отправить данные, воспользуйтесь функцией `CFSocketSendData`:

```
char joke[] = "Why did the chicken cross the road?";

kCFSocketError err = CFSocketSendData(mySocket, joke,
(strlen(joke)+1), 10);
```

Затем вы можете проверить код ошибки, чтобы понять, успешно ли были отправлены данные:

```
if (err == kCFSocketSuccess) {
  /* Успех */
}
```

В библиотеке CFNetwork используются следующие коды ошибок:

- `kCFSocketSuccess` — операция выполнена успешно;
- `kCFSocketError` — операция выполнена неудачно;
- `kCFSocketTimeout` — время выполнения операции истекло.

Обратные вызовы

Вы можете определить множество конкретных событий для инициирования обратных вызовов, например, поступление данных или получение новых подключений. Это позволит вам написать программное обеспечение, которое не требует блокировки или зацикливания для определения состояния сокета. CFNetwork для всех функций обратного вызова использует стандартный шаблон функции обратного вызова, предоставляя соответствующие этому типу обратного вызова данные. Вот прототип для данной функции:

```
typedef void (*CFSocketCallBack) (
    CFSocketRef s,
    CFSocketCallBackType callbackType,
    CFDataRef address,
    const void *data,
    void *info
);
```

Каждый обратный вызов сопровождает следующая информация. В зависимости от отправляемого типа обратного вызова часть информации может отличаться.

- `CFSocketRef s` — `CFSocket`, относящийся к произошедшему событию. Он позволяет вашей функции обратного вызова поддерживать несколько сокетов.
- `CFSocketCallBackType callbackType` — перечислимое значение для обратного вызова, определяющее тип произошедшего события. См. список типов обратного вызова ранее.
- `CFDataRef address` — объект `CFData`, содержащий низкоуровневую информацию `sockaddr`. Его можно использовать для получения удаленного адреса, к которому подключен сокет. Он предоставляется только во время обратных вызовов на установку соединения и отправку/прием данных.
- `const void *data` — указатель на специальные данные, относящиеся к обратному вызову. Для события отправки/приема данных (data event) передается объект `CFData`, содержащий полученные данные. Для события

согласия на установку подключения (accept event) предоставляется указатель на `CFSocketNativeHandle`, указывающий на встроенный объект сокета. Для события подключения (connect event) предоставляется указатель на код ошибки `SInt32`.

- `void *info` — указатель, предоставляемый для структуры `CFSocketContext`, сопоставленный сокету. Он содержит любые задаваемые пользователем данные, которые вы сопоставили сокету.

CFSocketContext

Поскольку `CFSockets` может работать в цикле в фоновом режиме, то отслеживание данных, связанных с каждым подключением, может стать нетривиальной задачей. Например, если вы пишете механизм поиска, то, возможно, создаете сотни подключений к различным Web-серверами и нуждаетесь в понимании того, какое подключение сопоставлено каким событиям обратного вызова. Объект `CFSocketContext` позволяет вам связать указатель на любую подобную информацию со структурой сокета, так чтобы она была доступна всякий раз, когда инициируется обратный вызов.

Вот прототип контекстной структуры:

```
struct CFSocketContext {
    CFIndex version;
    void *info;
    CFAllocatorRetainCallBack retain;
    CFAllocatorReleaseCallBack release;
    CFAllocatorCopyDescriptionCallBack copyDescription;
};
typedef struct CFSocketContext CFSocketContext;
```

Здесь:

- `CFIndex version` — номер версии структуры. Apple требует, чтобы это был `0`;
- `void *info` — указатель на задаваемые пользователем данные вашего приложения, которые будут сопоставлены объекту `CFSocket` при его создании. Этот указатель будет передаваться в списке аргументов всех обратных вызовов, инициированных объектом `CFSocket`;
- `CFAllocatorRetainCallBack retain` — необязательный обратный вызов, используемый при удержании контекста. Если нет необходимости в обратном вызове, то используйте значение `NULL`;

- `CFAllocatorReleaseCallBack release` — необязательный обратный вызов, используемый при освобождении контекста. Если нет необходимости в обратном вызове, то используйте значение `NULL`;
- `CFAllocatorCopyDescriptionCallBack copyDescription` — необязательный обратный вызов, вызываемый при копировании объекта в другой контекст. Если нет необходимости в обратном вызове, то используйте значение `NULL`.

Вот пример контекста сокета:

```
char joke[] = "Why did the chicken cross the road?";
CFSocketContext CTX = { 0, joke, NULL, NULL, NULL };
```

При выполнении действий с сокетом вы можете получить контекст сокета посредством вызова `CFSocketGetContext`. Эта функция копирует содержимое контекста сокета в локальную структуру, предоставляемую вызывающим:

```
CFSocketContext localCTX;
CFSocketGetContext(mySocket, &localCTX);
```

Потоки сокетов

Потоки сокетов (socket streams) предоставляют простой интерфейс для чтения данных из сокета и записи в него. Каждый сокет может читать поток или писать в него, позволяя тем самым осуществлять синхронное или асинхронное взаимодействие. Потоки инкапсулируют большую часть работы, необходимой для чтения и записи потоков байтов, и замещают стандартные функции кодов ошибок `send()` и `recv()`, используемые в C. С сокетами используются два различных объекта потока: `CFReadStream` и `CFWriteStream`.

Потоки чтения

Специальный набор функций `CFReadStream` позволяет осуществлять простые операции чтения на сокете. Практически так же, как и в C, используется буфер чтения, в котором зациклен поток чтения до тех пор, пока не будет прочитано требуемое количество байтов:

```
char buffer[1024];
CFIndex bytes_recvd;
int recv_len = 0;
```

```
memset(buffer, 0, sizeof(buffer));
while (!strchr(buffer, '\n') && recv_len < sizeof(buffer)) {
  bytes_recvd = CFReadStreamRead(readStream, buffer + recv_len,
        sizeof(buffer) -; recv_len);
  if (bytes_recvd < 0) {
     /* Произошла ошибка. Закрываем сокет и возвращаемся. */
  }
  recv_len += bytes_recvd;
}
```

Далее приведен список полезных функций `CFReadStream`:

- `CFReadStreamOpen` открывает, а `CFReadStreamClose` закрывает поток чтения. Эти функции выделяют и освобождают ресурсы, необходимые для выполнения потокового чтения. Поток должен быть открыт до того, как он будет прикреплен к сокету.
- `CFReadStreamRead` — непосредственно функция чтения потока. Эта функция выполняет считывание из прикрепленного сокета в предоставленный буфер, возвращая количество считанных байтов. Как и традиционные сокеты, чтение можно зациклить, пока не будет получено требуемое количество байтов.
- `CFReadStreamGetBuffer` возвращает указатель на внутренний буфер непрочитанных данных потока чтения, позволяя реализующему осуществлять доступ к буферу необработанных данных напрямую.
- `CFReadStreamGetStatus` возвращает текущее состояние потока чтения. Возможные состояния потока:
  - `kCFStreamStatusNotOpen` — поток чтения не открыт;
  - `kCFStreamStatusOpening` — поток чтения открывается для считывания;
  - `kCFStreamStatusOpen` — поток чтения открыт и готов;
  - `kCFStreamReading` — в текущий момент осуществляется считывание потока;
  - `kCFStreamStatusAtEnd` — больше нет данных для считывания из потока;
  - `kCFStreamStatusClosed` — поток чтения закрыт;
  - `kFStreamStatusError` — в потоке чтения произошла ошибка.

- `CFReadStreamHasBytesAvailable` возвращает булево значение, указывающее на то, готовы ли входные данные для считывания без блокирования. Эту функцию можно использовать для периодического опроса сокета на предмет наличия данных, хотя при использовании цикла в этом нет никакой необходимости.
- `CFReadStreamScheduleWithRunLoop`, `CFReadStreamUnscheduleFromRunLoop` используются для управления включением потока в цикл обработки. Помещенный в цикл обработки поток вызывается при наступлении определенных событий, таких как открытие, ошибки, а также появление доступных для чтения данных. Поток можно включить в несколько циклов обработки с использованием различных режимов.
- `CFReadStreamSetClient` назначает клиента для получения обратного вызова для потока, пока он находится в цикле обработки.

Если сокет уже существует, то функция `CFStreamCreatePairWithSocket` может автоматически инициализировать поток чтения и записи:

```
/* Встроенный сокет, используемый для различных операций */
CFReadStreamRef readStream;
CFWriteStreamRef writeStream;

CFSocketNativeHandle sock = *(CFSocketNativeHandle *) data;
CFStreamCreatePairWithSocket(kCFAllocatorDefault, sock,
                             &readStream, &writeStream);
```

Чтобы включить обработку потока чтения в главный цикл, вызовите его функцию планировщика (scheduler function). В результате поток чтения перейдет в фоновый режим обработки, и активируется следующая функция обратного вызова:

```
CFReadStreamScheduleWithRunLoop(readStream, CFRunLoopGetCurrent(),
                                kCFRunLoopCommonModes);
```

Когда поток чтения войдет в цикл обработки, функции обратного вызова клиента будут вызываться всякий раз, когда будут происходить определенные события:

```
typedef void (*CFReadStreamClientCallBack) (
    CFReadStreamRef stream,
    CFStreamEventType eventType,
    void *clientCallBackInfo
);
```

Аналогично функции обратного вызова `CFSocket` обратные вызовы потока предоставляют информацию о типе события и относящихся к этому событию данных. Обратный вызов можно вызывать для следующих событий:

- `kCFStreamEventNone` — неопределенное событие;
- `kCFStreamEventOpenCompleted` — поток был успешно открыт и готов для дальнейших операций;
- `kCStreamEventHasBytesAvailable` — имеются данные для потока для считывания;
- `kCFStreamEventErrorOccurred` — в потоке чтения произошла ошибка;
- `kCFStreamEventEndEncountered` — был достигнут конец файла, в потоке больше нет данных для чтения.

Потоки записи

Дополнением к потоку чтения является `CFWriteStream`. Этот поток создан для записи и управляет отправкой данных посредством `CFSocket`. Вот пример его использования:

```
char data[] = "Beware the Jabberwocky.\n";
CFIndex bytes_sent = 0;
int send_len = 0;

if (CFWriteStreamCanAcceptBytes(writeStream)) {
  bytes_sent = CFWriteStreamWrite(writeStream, data + send_len,
                                  (strlen(data)+1) - send_len);
  if (bytes_sent < 0) {
    /* Произошла ошибка отправки. Закрываем сокет и возвращаемся. */
  }
  send_len += bytes_sent;
}
```

Далее приведен список полезных функций `CFWriteStream`.

- `CFWriteStreamOpen` открывает, а `CFWriteStreamClose` закрывает поток записи. Эти функции выделяют и освобождают ресурсы, необходимые для выполнения потоковой записи. Поток должен быть открыт до того, как он будет прикреплен к сокету.

- `CFWriteStreamWrite` — непосредственно функция записи потока записи. Эта функция выполняет запись из источника данных в соответствующий сокет, возвращая количество отправленных байтов. Как и традиционные функции `send`*`XXX`*`()`, выполняемые на сокетах C, запись можно зациклить до тех пор, пока не будет отправлено требуемое количество байтов.
- `CFWriteStreamGetStatus` возвращает текущее состояние потока записи. Возможные состояния потока:
  - `kCFStreamStatusNotOpen` — поток записи не открыт;
  - `kCFStreamStatusOpening` — поток записи открывается для записи;
  - `kCFStreamStatusOpen` — поток записи открыт и готов;
  - `kCFStreamWriting` — в текущий момент осуществляется запись в поток;
  - `kCFStreamStatusAtEnd` — больше нет данных для записи в поток;
  - `kCFStreamStatusClosed` — поток записи закрыт;
  - `kFStreamStatusError` — в потоке записи произошла ошибка.
- `CFReadStreamCanAcceptBytes` возвращает булево значение, указывающее, можно ли писать в поток.
- `CFWriteStreamSheduleWithRunLoop`, `CFWriteStreamUnscheduleFromRunLoop` используются для управления включением потока в цикл обработки. Помещенный в цикл обработки поток вызывается при наступлении определенных событий, таких как открытие, ошибки, а также окончание записи данных. Поток можно включить в несколько циклов обработки с помощью различных режимов.
- `CFWriteStreamSetClient` назначает клиента для получения обратного вызова для потока, пока он находится в цикле обработки.

Чтобы включить обработку потока записи в главный цикл обработки, вызовите его функцию планировщика (scheduler function). В результате поток чтения перейдет в фоновый режим обработки, и активируется следующая функция обратного вызова:

```
CFWriteStreamScheduleWithRunLoop(writeStream, CFRunLoopGetCurrent(),
                                 kCFRunLoopCommonModes);
```

Когда поток записи войдет в цикл обработки, функции обратного вызова клиента будут вызываться всякий раз, когда будут происходить определенные события:

```
typedef void (*CFWriteStreamClientCallBack) (
    CFWriteStreamRef stream,
```

```
    CFStreamEventType eventType,
    void *clientCallBackInfo
);
```

Аналогично функции обратного вызова `CFSocket` обратные вызовы потока предоставляют информацию о типе события и относящихся к этому событию данных. Обратный вызов можно вызывать для следующих событий:

- `kCFStreamEventNone` — неопределенное событие;
- `kCFStreamEventOpenCompleted` — поток был успешно открыт и готов для дальнейших операций;
- `kCStreamEventCanAcceptBytes` — поток готов к записи;
- `kCFStreamEventErrorOccurred` — в потоке записи произошла ошибка;
- `kCFStreamEventEndEncountered` — был достигнут конец файла, в потоке больше нет данных.

Пример с *CFSocket*: сервер анекдотов

В этом примере будут применены ваши знания структур `CFSocket` и `CFSocketContext` и потоков для построения TCP-серверной программы, которая будет сообщать окончание любого запрашиваемого у нее анекдота. Окончания анекдотов вы будете хранить в `CFSocketContext` на стороне сервера и предоставлять ответ клиенту. Вы, несомненно, обратите внимание на потрясающую схожесть со стандартным программированием сокетов на C, однако пример сервера продемонстрирует все возможности цикла обработки библиотеки CFNetwork, которые существенно облегчают управление сокетами.

Приведенный далее код настраивает сокет сервера. Код сопровождается построчными комментариями:

```
/* Сокет сервера */
CFSocketRef TCPServer;
#define PORT 2048

/* Окончание нашего анекдота */
char punchline[] = "To get to the other side!";

/* Используется setsockopt */
int yes = 1;
```

```
/* Создаем контекст нашего сокета; это связывает окончание */
/* анекдота с сокетом */
CFSocketContext CTX = {0, punchline, NULL, NULL, NULL};

/* Создаем сокет сервера как сокет TCP IPv4 и задаем обратный вызов */
/* для вызовов низкоуровневой функции сокета accept() */
TCPServer = CFSocketCreate(kCFAllocatorDefault, PF_INET, SOCK_STREAM,
                           IPPROTO_TCP, kCFSocketAcceptCallBack,
                          (CFSocketCallBack)&AcceptCallback, &CTX);
if (TCPServer == NULL)
    return NULL;

/* Повторно используем локальные адреса, */
/* если они все еще находятся в TIME_WAIT */
setsockopt(CFSocketGetNative(TCPSocket), SOL_SOCKET, SO_REUSEADDR,
    (void *)&yes, sizeof(yes));

/* Устанавливаем порт и адрес, которые мы собираемся слушать */
struct sockaddr_in addr;
memset(&addr, 0, sizeof(addr));
addr.sin_len = sizeof(addr);
addr.sin_family = AF_INET;
addr.sin_port = htons(PORT);
addr.sin_addr.s_addr = htonl(INADDR_ANY);
NSData *address = [ NSData dataWithBytes: &addr, length:sizeof(addr) ];
if (CFSocketSetAddress(TCPServer, (CFDataRef)address) !=
  kCFSocketSuccess) {
    fprintf(stderr, "CFSocketSetAddress() failed\n");
    CFRelease(TCPServer);
    return NULL;
}

CFRunLoopSourceRef sourceRef =
    CFSocketCreateRunLoopSource(kCFAllocatorDefault, TCPServer, 0);
```

```
CFRunLoopAddSource(CFRunLoopGetCurrent(), sourceRef,
                   kCFRunLoopCommonModes);
CFRelease(sourceRef);

/* Продолжаем жить нашей жизнью, а не ждем входящие подключения. */
```

Теперь наш ожидающий сокет установлен, а функция обратного вызова будет вызываться всякий раз, когда будет устанавливаться новое подключение. Приведенная далее функция обратного вызова является одним из примеров того, как можно обрабатывать входящее подключение. В данном примере функция обратного вызова создает и объединяет в пару поток чтения и записи с сокетом. Потом она ждет поступления анекдота с другого конца, а затем отправляет окончание анекдота:

```
static void AcceptCallBack(CFSocketRef socket,
                           CFSocketCallBackType type,
                           CFDataRef address,
                           const void *data,
                           void *info)
{
    CFReadStreamRef readStream = NULL;
    CFWriteStreamRef writeStream = NULL;
    CFIndex bytes;
    UInt8 buffer[128];
    UInt8 recv_len = 0, send_len = 0;

    /* Встроенный сокет, используемый для различных операций */
    CFSocketNativeHandle sock = *(CFSocketNativeHandle *) data;

    /* Окончание анекдота, которое мы сохранили в контексте сокета */
    char *punchline = info;

    /* Создаем потоки чтения и записи для сокета */
    CFStreamCreatePairWithSocket(kCFAllocatorDefault, sock,
        &readStream, &writeStream);

    if (!readStream || !writeSream) {
        close(sock);
```

```
        fprintf(stderr, "CFStreamCreatePairWithSocket() failed\n");
        return;
    }

    /* Ожидаем завершения отправки анекдота клиентом; ожидаем newline */
    memset(buffer, 0, sizeof(buffer));
    while (!strchr(buffer, '\n') && recv_len < sizeof(buffer)) {
        bytes = CFReadStreamRead(readStream, buffer + recv_len,
                                 sizeof(buffer)-recv_len);
        if (bytes < 0) {
            fprintf(stderr, "CFReadStreamRead() failed\n");
            close(sock);
            return;
        }
        recv_len += bytes;
    }

    /* Отправляет окончание анекдота */
    while (send_len < (strlen(punchline+1))) {
    if (CFWriteStreamCanAcceptBytes(writeStream)) {
      bytes = CFWriteStreamWrite(writeStream, punchline + send_len,
                                 (strlen(punchline)+1) - send_len);
      if (bytes < 0) {
        fprintf(stderr, "CFWriteStreamWrite() failed\n");
        close(sock);
        return;
      }
      send_len += bytes;
    }
    close(sock);
    return;
}
```

Для дальнейшего изучения

Взгляните на CFSocket.h, CFStream.h и CFSocketContext.h. Вы найдете их в заголовках Core Foundation вашего SDK в папке /Developer/Platforms/iPhoneOS.platform.

Интерфейсы *CFHTTP* и *CFFTP*

Библиотека CFNetwork предоставляет простой API для осуществления HTPP- и FTP-запросов. Эти запросы вы можете делать посредством интерфейсов `CFHTTP` и `CFFTP`. Вместо того чтобы программировать в вашем приложении возможность разговаривать на определенном протоколе, эти API позволяют библиотеке делать за вас всю черную работу, возвращая необработанный протокол в последовательной форме, которую вы можете затем отправить посредством потока записи, подключенного к требуемому вам узлу, или непосредственно подключить к потоку для открытия.

Хотя многие классы Cocoa, например, такие как `NString` и `NSData`, позволяют вам инициализировать объекты с содержимым URL, интерфейсы `CFHTTP` и `CFFTP`, как вы увидите в следующем примере, предоставляют более детальное управление уровнем протокола.

CFHTTP

Для создания HTTP-запроса можно использовать API `CFHTTP`. Это позволит вам легко делать GET, HEAD, PUT, POST и большинство других стандартных HTTP-запросов. Создание запроса является трехступенчатым процессом: создание объекта запроса, описание сообщения и заголовков HTTP-запроса, сериализация сообщения в необработанный протокол. Тело сообщения, которое может содержать данные для отправки, как правило, содержат только HTTP-запросы POST. Все остальные запросы используют пустое тело, внедряя параметры запроса в заголовки.

В приведенном далее примере создается запрос GET протокола HTTP/1.1 с указанием URL: **http://www.oreilly.com** и заданием заголовка `Connection`, указывающего удаленному концу закрыть подключение после завершения отправки данных:

```
CFStringRef requestHeader = CFSTR("Connection");
CFStringRef requestHeaderValue = CFSTR("close");
CFStringRef requestBody = CFSTR("");

CFStringRef url =
  CFSTR("http://www.oreilly.com">http://www.oreilly.com");
CFStringRef requestMethod = CFSTR("GET");
CFURLRef requestURL = CFURLCreateWithString(kCFAllocatorDefault,
                                            url, NULL);
```

```
CFHTTPMessageRef request =
  CFHTTPMessageCreateRequest(kCFAllocatorDefault,
    requestMethod, requestURL, kCFHTTPVersion1_1);
CFHTTPMessageSetBody(request, requestBody);
CFHTTPMessageSetHeaderFieldValue(request, requestHeader,
                                 requestHeaderValue);

CFDataRef serializedRequest =
  CFHTTPMessageCopySerializedMessage(request);
```

Получаемый в результате указатель на структуру `CFData` адресует необработанный вывод протокола HTTP, который затем через поток записи направляется на сервер назначения. В приведенном далее примере создается HTPP-запрос GET и открывается посредством потока чтения. По мере поступления данных для их обработки будут вызываться обратные вызовы этого потока чтения:

```
int makeRequest(const char *requestURL)
{
    CFReadStream readStream;
    CFHTTPMessageRef request;
    CFStreamClientContext CTX = { 0, NULL, NULL, NULL, NULL };

    NSString* requestURLString = [ [ NSString alloc ] initWithCString:
        requestURL ];
    NSURL url = [ NSURL URLWithString: requestURLString ];

    CFStringRef requestMessage = CFSTR("");
    request = CFHTTPMessageCreateRequest(kCFAllocatorDefault,
                                         CFSTR("GET"),
        (CFURLRef) url, kCFHTTPVersion1_1);
    if (!request) {
        return -1;
    }
    CFHTTPMessageSetBody(request, (CFDataRef) requestMessage);
    readStream = CFReadStreamCreateForHTTPRequest(kCFAllocatorDefault,
                   request);
    CFRelease(request);
```

```
    if (!readStream) {
        return -1;
    }
    if (!CFReadStreamSetClient(readStream, kCFStreamEventOpenCompleted |
                               kCFStreamEventHasBytesAvailable |
                               kCFStreamEventEndEncountered |
                               kCFStreamEventErrorOccurred,
                               ReadCallBack, &CTX))
    {
      CFRelease(readStream);
      return -1; }

    /* Добавляем в цикл обработки */
    CFReadStreamScheduleWithRunLoop(readStream, CFRunLoopGetCurrent(),
        kCFRunLoopCommonModes);

    if (!CFReadStreamOpen(readStream)) {
        CFReadStreamSetClient(readStream, 0, NULL, NULL);
        CFReadStreamUnscheduleFromRunLoop(readStream,
            CFRunLoopGetCurrent(), kCFRunLoopCommonModes);
        CFRelease(readStream);
        return -1;
    }

    return 0;
}
```

CFFTP

API `CFFTP` схоже с API `CFHTTP` и опирается при передаче FTP-данных на потоки чтения. Чтобы создать FTP-запрос, воспользуйтесь функцией `CFReadStreamCreateWithFTPURL`, как показано далее.

Тем самым будет создан исходный поток чтения на FTP-сервер:

```
CFStringRef url = CFSTR("ftp://ftp.somedomain.com/file.txt");
CFURLRef requestURL = CFURLCreateWithString(kCFAllocatorDefault,
   url, NULL);
```

```
CFReadStreamRef readStream = CFReadStreamCreateWithFTPURL(
    kCFAllocatorDefault, requestURL);
```

После создания потока чтения вы можете связать с ним функцию обратного вызова так, чтобы функция чтения вызывалась, когда данные будут готовы:

```
CFReadStreamSetClient(readStream,
                      kCFtreamEventHasBytesAvailable,
                      readCallBack,
                      NULL);
```

Функция обратного вызова будет вызываться всякий раз, когда будут доступны данные, и сможет считывать данные с потока:

```
void readCallBack(CFReadStreamRef stream,
                  CFStreamEventType eventType,
                  void *clientCallBackInfo)
{
    char buffer[1024];
    CFIndex bytes_recvd;
    int recv_len = 0;

    while(recv_len < total_size && recv_len < sizeof(buffer)) {
        bytes_recvd = CFReadStreamRead(stream, buffer + recv_len,
                                       sizeof(buffer) - recv_len);
        /* Записываем байты в вывод или в файл */
    }
    recv_len += bytes_recvd;
}
```

Теперь вы можете поместить этот запрос в главный цикл обработки вашей программы. Как только в потоке чтения будут появляться данные, ваш обратный вызов чтения будет вызываться и продолжать обрабатывать данные до тех пор, пока не будет закрыто подключение или не будет достигнут конец файла:

```
CFReadStreamScheduleWithRunLoop(readStream,
    CFRunLoopGetCurrent(), kCFRunLoopCommonModes);
```

Для дальнейшего изучения

Взгляните на CFFTPStream.h, CFHTTPStream.h, и CFNetwork.h. Вы найдете их в заголовках Core Foundation вашего SDK в папке /Developer/Platforms/iPhoneOS.platform.

Глава 8

Определение местоположения: Core Location

Библиотека Core Location предоставляет доступ к системам определения географического местоположения, имеющимся в iPhone. Таковыми системами являются встроенный GPS (integrated GPS), определение местоположения на базе Wi-Fi (WiFi-based location positioning) и определение местоположения на базе показаний вышек (tower triangulation). iPhone — это устройство, которое очень многое знает о местоположении и даже хранит в кэше GPS-координаты близлежащих вышек. С помощью библиотеки Core Location вы сможете получать для устройства координаты долготы и широты, высоту над уровнем моря и другие не менее важные данные. Core Location использует подобие потокового уведомления, поэтому ваше приложение будет получать обновления сразу же, как только GPS обнаружит более точные координаты местоположения.

Apple добавила в интерфейс Core Location базовый уровень безопасности, и когда ваше приложение попытается воспользоваться этой библиотекой, у пользователя попросят разрешение на доступ к его текущему местоположению. Если пользователь запретит вашему приложению использовать Core Location, то оно должно будет предоставить функциональность, не использующую эту информацию, либо запросить у пользователя почтовый индекс или другую подобную информацию. Помимо этого, Apple внедрила во встроенное программное обеспечение iPhone черный список (blacklist). Всякий раз при попытке осуществления поиска iPhone опрашивает этот список на серверах Apple и уничтожает любые приложения, попавшие в этот список.

Процесс поиска местоположения достаточно энергозатратный, поскольку при этом опрашиваются спутники GPS, местные вышки и точки доступа WiFi. Поэтому будьте сдержанны в своем применении Core Location, чтобы не разрядить аккумулятор пользователя.

Чтобы воспользоваться библиотекой Core Location, вам потребуется добавить ее в ваш проект Xcode. Для этого щелкните правой кнопкой мыши по папке Frameworks в вашем проекте Xcode, а затем в появившемся контекстном меню выберите команду **Add | Existing Frameworks**. Перейдите к папке CoreLocation.framework, а затем нажмите кнопку **Add**.

> ***Внимание!***
>
> На момент написания этой книги политикой AppStore было запрещено размещение в этом электронном магазине приложений, предоставляющих пошаговые инструкции по маршруту движения, и средств воздушной навигации. Убедитесь, что, используя информацию о местоположении, вы не нарушаете условий вашего соглашения с Apple.

Менеджер Core Location

Все функции Core Location выполняются посредством менеджера Core Location. Задача этого менеджера — выступать в роли интерфейса для вашего приложения. Вы порождаете менеджер для настройки и выполнения запросов на определение местоположения, а также для выключения GPS. Менеджер порождается как объект `CLLocationManager`. Ответы на запросы на определение вашего местоположения приходят не мгновенно. Когда обнаруживается информация (или обновление) о местонахождении пользователя, ваше приложение получает соответствующее уведомление. Эта информация периодически отправляется делегату менеджера, который должен реализовывать протокол `CLLocationManagerDelegate`. Последующие обновления местоположения устройства отправляются тогда, когда Core Location получит более точную информацию, или же когда устройство будет перемещено на расстояние, превышающее пороговую величину, заданную программистом.

В листинге 8.1 вы создадите собственный класс, который будет выступать в роли класса запроса. Класс запроса будет выполнять функции делегата, вызывать класс менеджера Core Location, получать уведомления и, в конце концов, освобождать ресурсы менеджера по окончании работы с ним.

Листинг 8.1. Описание класса `MyCLQuery`

```
@interface MyCLQuery : NSObject
{
    CLLocationManager *manager;
}
- (void)startQuery;
- (void)stopQuery;
- (void)locationManager:(CLLocationManager *)locationManager
    didUpdateToLocation:(CLLocation *)newLocation
    fromLocation:(CLLocation *)oldLocation;
```

Класс `MyCLQuery` в примере использует три метода. Метод `startQuery` задает объект `CLLocationManager` и начинает процесс определения местоположения. После своей активации начинает работу в фоновом режиме менеджер местоположения для определения вашего текущего местоположения. Как только координаты местоположения будут получены или оно будет изменено, уведомляется метод `didUpdateToLocation`. Этот метод ответственен за передачу вашему приложению информации о новом местоположении. Наконец, когда вы завершите использование Core Location, необходимо вызвать метод `stopQuery`, чтобы отключить запросы на обновление.

Параметры запроса

Чтобы сообщить Core Location, какого рода информацию необходимо получать, вам придется предоставить некоторые параметры запроса. Эти параметры касаются требуемой степени точности и периодичности отправки обновлений вашему делегату. Например, приложению, предназначенному для поиска ближайшего ресторана, требуется менее специфическая информация о местоположении, нежели приложению геокешинга (geo-caching), которому потребуются особые данные о местоположении пользователя. Если ваше приложение будет использоваться при движении на высоких скоростях (например, при путешествии на автомобиле или в автобусе), то ему будет достаточно получать обновления каждые 30 метров, а приложению, предназначенному для использования при пешем перемещении, могут потребоваться более частые обновления координат местоположения.

Прежде чем приказать менеджеру начать процесс определения местоположения, необходимо задать, по крайней мере, три параметра.

- `delegate`. Этот объект будет получать обновления координат местоположения. В объекте-делегате необходимо корректно реализовать протокол `CLLocationManagerDelegate`. Для получения этих уведомлений от менеджера Core Location делегат будет уведомляться посредством метода `didUpdateToLocation`, как показано в листинге 8.1. В примерах этой главы в качестве делегата выступает объект `MyCLQuery`.
- `desiredAccuracy` указывает менеджеру местоположения степень точности, требуемой в вашем приложении. С целью экономии заряда аккумуляторов и уменьшения излишних обновлений используйте наименьшую степень точности, необходимую в вашем приложении. Различные степени точности достигаются с разной степенью легкости. Например, если в пределах досягаемости сигнала имеется только одна сотовая вышка, то Core Location сначала может передать вашему приложению очень широкий разброс координат. По мере дальнейшего получения Core Location более точной информации с помощью средств определения местоположения на базе Wi-Fi и GPS она будет передавать вашему приложению более точные координаты местоположения. Задать частоту и точность определения местоположения пользователя можно с помощью следующих параметров:
  - `kCLLocationAccuracyBest` — наилучшая возможная точность;
  - `kCLLocationAccuracyNearestTenMeters` — точность в пределах 10 метров (30 футов);
  - `kCLLocationAccuracyHundredMeters` — точность в пределах 100 метров (300 футов);
  - `kCLLocationAccuracyKilometer` — точность в пределах 1 километра (около 0,6 мили);
  - `kCLLocationAccuracyThreeKilometers` — точность в пределах 3 километров (около полутора миль).

  Примечание

  Один метр равен 1,09 ярда. Один километр равен 0,62 мили.

- `distanceFilter` задает минимальное расстояние (в метрах), на которое должно быть перемещено устройство, прежде чем обновление будет отправлено делегату. Дальнейшие обновления будут отправляться только

после того, как устройство будет перемещено на расстояние, большее или равное этому значению, или же если будет увеличена степень точности. Чтобы отправлять обновления при любом перемещении, используйте в качестве значения этого параметра `kCLDistanceFilterNone`.

Выполнение запроса

После того как вы определили, какого рода запрос Core Location требуется вашему приложению, вы можете породить менеджер и назначить ему параметры вашего запроса. В приведенном далее примере для настройки запроса и вызова Core Location используется метод `startQuery`. Код запрашивает наилучшую возможную точность и заказывает постоянные обновления при любом перемещении. Наконец, он вызывает метод `startUpdatingLocation` менеджера Core Location, который инициирует процесс определения местоположения и отправки уведомлений:

```
- (void) startQuery
{
    manager = [[ CLLocationManager alloc ] init ];

    manager.delegate = self;

    manager.desiredAccuracy = kCLLocationAccuracyBest;
    manager.distanceFilter = kCLDistanceFilterNone;

    NSLog(@"Core Location updates started");
    [ manager startUpdatingLocation ];
}
```

Получение обновлений

Роль делегата в нашем примере играет объект `MyCLQuery`, поэтому вам необходимо реализовать методы протокола делегата. Менеджер Core Location предоставляет два результата, указывающих как на новое, так и на старое местоположение, облегчая тем самым распознавание факта перемещения. От обоих объектов `CLLocation` можно получить следующую информацию.

- *Долгота и широта.* Долгота (longitude) и широта (latitude) считываются посредством структуры `coordinate`, доступной из объекта `CLLocation`.

Эта структура содержит члены `latitude` и `longitude`, представленные значениями с плавающей точкой двойной точности (double floating point); они приводятся к типу данных `CLLocationDegrees`.

- *Высота над уровнем моря.* Свойство `altitude` считывает высоту над уровнем моря (altitude) в метрах. Это значение может быть положительным или отрицательным, определяя количество метров выше или ниже уровня моря соответственно. Это значение имеет тип данных с плавающей точкой двойной точности к `CLLocationDistance`.
- *Точность.* Точность (accuracy) по горизонтали и по вертикали имеет тип данных `CLLocationAccuracy` (с плавающей запятой двойной точности). Эти значения, доступные посредством свойств `horizontalAccuracy` и `verticalAccuracy`, обозначают точность (в метрах) предоставляемых координат и высоты над уровнем моря. Вам не всегда гарантирована запрашиваемая вами степень точности, поэтому необходимо проверять ее для определения действительного радиуса возвращаемого местоположения. По мере улучшения степени точности эти значения также будут обновляться с отправкой последующих уведомлений.
- *Временная метка.* Обновления отправляются периодически по мере улучшения GPS или когда устройство перемещается на заданное расстояние. Если ваше приложение работает в режиме реального времени или максимально приближенном к нему, то проверить, что возвращаемые данные являются самыми свежими, можно с помощью свойства `timestamp`, являющегося объектом `NSDate`, содержащим временную метку (timestamp) запроса.

  Поскольку Core Location для получения координат местоположения использует различные возможности, то вероятна ситуация, когда более новые координаты, возвращаемые в приложение, будут иметь более старую временную метку, чем предыдущие координаты. Например, запрос на глобальное местоопределение может занять несколько секунд, а определение местоположения на базе Wi-Fi — менее одной секунды. В результате более точные координаты, возвращаемые спутником, будут иметь более позднюю временную метку. Лучший вариант действий в этой ситуации — это принять более новую информацию, если она является более точной, и предоставить пользователю визуальный индикатор обоих местоположений.
- *Описание.* Core Location также предоставляет объект `NSString`, описывающий общее местоположение. Вы можете получить доступ к нему посредством свойства `description`. Apple не гарантирует неизменности

формата этого свойства, поэтому не стоит разбирать данные, основываясь на этом описании. Однако его можно отображать пользователю или использовать в целях отладки.

Приводимый далее пример получает уведомления, отправляемые менеджером Core Location, и выводит на консоль выходную информацию:

```
- (void)locationManager:(CLLocationManager *)locationManager
didUpdateToLocation:(CLLocation *)newLocation
fromLocation:(CLLocation *)oldLocation
{
    NSLog(@"Old Coordinates: %+.6f, %+.6f\n",
        oldLocation.coordinate.latitude,
        oldLocation.coordinate.longitude);
    NSLog(@"New Coordinates: %+.6f, %+.6f\n",
        newLocation.coordinate.latitude,
        newLocation.coordinate.longitude);
    NSLog(@"Altitude: %.6f\n", newLocation.altitude);
    NSLog(@"Description: %@\n", [ newLocation description ]);

    NSDate *timestamp = newLocation.timestamp;
    NSTimeInterval age = [ timestamp timeIntervalSinceNow ];
    NSLog(@"Notification age: %.4f\n", age);
}
```

Завершение запроса

Если вашему приложению достаточно полученной информации о местоположении, обязательно остановите менеджер, чтобы избежать разрядки аккумуляторов. Тем самым вы остановите и процесс отправки вашему приложению обновлений. Сделать это можно с помощью простого вызова метода `stopUpdatingLocation` менеджера:

```
- (void) stopQuery
{
    [ manager stopUpdatingLocation ];
    [ manager release ];
}
```

Обработка ошибок

Если определить текущее местоположение невозможно, или пользователь запретил приложению использовать его текущее местоположение, то может возникнуть ошибка. Если происходит что-либо подобное, то метод `didFailWithError` уведомляет делегата о сбое:

```
- (void) locationManager:(CLLocationManager *)locationManager
        didFailWithError:(NSError *)error
{
    NSLog(@"Core Location failed with error: %@\n", [ error code ]);
}
```

Метод `didFailWithError` предоставляет объект `NSError`, который является стандартным объектом Cocoa, используемым для информирования об ошибках. Код ошибки, возвращаемый вместе с объектом `NSError`, определяет тип произошедшего сбоя:

- `kCLErrorLocationUnknown` — невозможно определить местоположение. Если происходит эта ошибка, то Core Location продолжит попытки получить местоположение в фоновом режиме и отправит обновление, как только такая попытка окажется успешной. Если ваше приложение решит приостановить запрос, убедитесь в том, что оно воспользовалось методом `stopUpdatingLocation` менеджера, чтобы отключить Core Location.

  Если это происходит, то лучше всего уведомить пользователя о том, что ваше приложение столкнулось с трудностями при определении его местоположения, а не создавать у него ощущение, что процесс поиска продолжается. Проще всего это сделать с помощью индикатора активности, рассмотренного в *главе 10*;

- `kCLErrorDenied` — пользователь отказал вашему приложению в доступе к Core Location. Если это произошло, то ваше приложение должно быть готово либо к продолжению работы без информации о местоположении, либо попросить пользователя вручную ввести, например, такие данные, как почтовый индекс.

Определение местоположения: WhereYouAt

Этот пример демонстрирует применение Core Location, запрашивая у объекта `CLLocationManager` текущее местоположение. Полученная информация будет предоставлена пользователю в тестовом представлении.

Вы можете скомпилировать это приложение, показанное в листингах 8.2—8.6, с помощью SDK, создав проект приложения WhereYouAt на базе представления (view-based application). Если вы хотите увидеть, как создавать все эти объекты с нуля, то удалите код Interface Builder.

Листинг 8.2. Прототипы делегата приложения WhereYouAt (WhereYouAtAppDelegate.h)

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

@class WhereYouAtViewController;

@interface WhereYouAtAppDelegate : NSObject <UIApplicationDelegate> {
    UIWindow *window;
    WhereYouAtViewController *viewController;
}

@property (nonatomic, retain) IBOutlet UIWindow *window;
@property (nonatomic, retain) IBOutlet WhereYouAtViewController
   *viewController;

@end
```

Листинг 8.3. Делегат приложения WhereYouAt (WhereYouAtAppDelegate.m)

```
#import "WhereYouAtAppDelegate.h"
#import "WhereYouAtViewController.h"

@implementation WhereYouAtAppDelegate

@synthesize window;
@synthesize viewController;

- (void)applicationDidFinishLaunching:(UIApplication *)application {
    CGRect screenBounds = [ [ UIScreen mainScreen ] bounds ];

    self.window = [ [ [ UIWindow alloc ] initWithFrame: screenBounds ]
        autorelease
    ];
```

```
    viewController = [ [ WhereYouAtViewController alloc ] init ];

    [ window addSubview: viewController.view ];
    [ window makeKeyAndVisible ];
}

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

@end
```

Листинг 8.4. Прототипы контроллера представлений WhereYouAt (WhereYouAtViewController.h)

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

@interface WhereYouAtViewController : UIViewController <CLLocationMana-
gerDelegate> {
    UITextView *textView;
    CLLocationManager *gps;
}

@end
```

Листинг 8.5. Контроллер представлений WhereYouAt (WhereYouAtViewController.m)

```
#import "WhereYouAtViewController.h"

@implementation WhereYouAtViewController

- (void)loadView {
    [ super loadView ];
    textView = [ [ UITextView alloc ] initWithFrame: self.view.frame ];
```

```
    self.view = textView;
}

- (void)viewDidLoad {
    textView.text = @"Where you at? ...";
    gps = [ [ CLLocationManager alloc ] init ];
    gps.delegate = self;
    gps.desiredAccuracy = kCLLocationAccuracyBest;
    gps.distanceFilter = kCLDistanceFilterNone;
    [ gps startUpdatingLocation ];
}

- (void)locationManager:(CLLocationManager *)locationManager
    didUpdateToLocation:(CLLocation *)newLocation
           fromLocation:(CLLocation *)oldLocation;
{
    textView.text = [ NSString stringWithFormat: @"You now at...\n\n"
        "Description: %@\n"
        "Coordinates: %f, %f\n"
        "Altitude: %f\n"
        "Updated: %@\n",
        newLocation.description,
        newLocation.coordinate.latitude,
        newLocation.coordinate.longitude,
        newLocation.altitude,
        newLocation.timestamp ];
}

- (void)didReceiveMemoryWarning {
    [ super didReceiveMemoryWarning ];
}

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

Листинг 8.6. Функция `main` для WhereYouAt (main.m)

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

int main(int argc, char *argv[]) {
    NSAutoreleasePool * pool = [ [ NSAutoreleasePool alloc ] init ];
    int retVal = UIApplicationMain(argc, argv, nil,
                                   @"WhereYouAtAppDelegate");
    [ pool release ];
    return retVal;
}
```

Как это работает

Приложение WhereYouAt является простым, но очень полезным примером. Вот как оно работает:

1. При своем порождении приложение уведомляет класс `WhereYouAtAppDelegate`, который, в свою очередь, создает окно и контроллер представлений.
2. Когда контроллер представлений загрузится, он создает объект `UITextView` и делает его активным представлением. Именно здесь пользователю будет отображаться информация о местоположении.
3. После загрузки представления метод `viewDidLoad` контроллера представлений создает экземпляр класса `CLLocationManager` и делает запрос.
4. Всякий раз, когда обнаруживается новая информация о местоположении, вызывается метод-делегат `didUpdateToLocation` с самой последней полученной информацией о местоположении. А она уже, в свою очередь, отображается пользователю.

Для дальнейшего изучения

Кладезем информации о запросах Core Location являются CLLocation.h и CLLocationManager.h. Вы найдете их в заголовках CoreLocation.framework вашего SDK в папке /Developer/Platforms/iPhoneOS.platform.

Глава 9

Библиотеки для работы с адресной книгой

Для управления адресной книгой iPhone SDK предоставляет две библиотеки: библиотеку доступа к низкоуровневым данным (AddressBook.framework) и пакет высокоуровневого пользовательского интерфейса (AddressBookUI.framework). Поскольку ваше приложение выполняется в рамках песочницы (sandbox), то вам запрещено напрямую взаимодействовать с базой данных SQLite, служащей основой для адресной книги. В этой главе вы узнаете, как запрашивать информацию о контактах с помощью вызовов функций языка C, и как работать с пользовательским интерфейсом для выбора контактов.

Если ваше приложение использует адресную книгу, то вам придется изучить имеющиеся в C функции доступа к данным, независимо от того, собираетесь ли вы или нет добавлять в ваше приложение пользовательский интерфейс адресной книги. Эти функции позволят вам считывать и записывать информацию о контактах. Поскольку вы имеете дело с личными данными другого человека, то очень важно до внесения каких-либо изменений в адресную книгу убедиться в том, что ваш код корректен и надежен. В отличие от библиотеки Core Location, у пользователя никогда не спрашивают разрешения на доступ к адресной книге. Однако в связи с возникающим в данном случае риском сохранения неприкосновенности личной жизни такое положение вещей может измениться в следующих версиях встроенного программного обеспечения iPhone. Использование библиотек для работы с адресной книгой для сбора личной информации о контактах пользователя нарушает соглашения Apple и, возможно, является незаконным во многих странах.

Чтобы воспользоваться библиотекой Address Book или библиотекой Address Book UI, вам потребуется добавить их в ваш проект Xcode. Для этого щелкните правой кнопкой мыши по папке Frameworks в вашем проекте, а затем в появившемся контекстном меню выберите команду **Add | Framework**. Перейдите к папке AddressBook.framework, а затем нажмите кнопку **Add**. Повторите эту же процедуру для папки AddressBookUI.framework.

> ***Внимание!***
>
> Чтобы найти библиотеки для работы с адресной книгой, вам, возможно, придется вручную перейти либо в папку /Developer/Platforms/iPhoneOS.platform, либо в папку /Developer/Platforms/iPhoneSimulator.platform и найти папку Frameworks в вашем SDK.

Доступ к адресной книге

Интерфейсами для доступа к адресной книге и к данным отдельных ее контактов являются функции на основе языка C, которые используют ссылки для передачи доступа к различным объектам адресной книги. Базовым объектом управления записями адресной книги является объект `ABRecord`. Он может представлять одного человека (`ABPerson`) или группу людей (`ABGroup`). Когда в пользовательском интерфейсе адресной книги выделяется какая-либо запись или же она возвращается из запроса средствами библиотеки, система возвращает указатель на `ABRecord`, представленный как `ABRecordRef`. В основном при взаимодействии с API адресной книги вы будете работать со ссылками `ABRecordRef`.

`ABRecordRef`, представляющий ссылку на ровно один контакт, имеет множество функций, используемых для чтения и записи информации. В этой главе вы узнаете о многих из них. Наиболее распространенными функциями, используемыми для взаимодействия с записями адресной книги, являются следующие.

- `ABRecordID ABRecordGetRecordID(ABRecordRef record);`

  Возвращает `ABRecordID` — 32-битное целое число, представляющее ID записи внутри базы данных. Это может оказаться полезным, когда вам требуется первичный ключ для отслеживания нескольких записей. Также контакт можно запросить по ID записи, о чем вы узнаете далее в этой главе.

- `ABRecordType ABRecordGetRecordType(ABRecordRef record);`

  Возвращает тип сущности, который представляет данная запись. Возможные значения `ABRecordType` могут относиться как к человеку (`kABPersonType`), так и к группе (`kABGroupType`).

- `CFStringRef ABRecordCopyCompositeName(ABRecordRef record);`

  Возвращает полное имя человека или группы. Типы данных `CFStringRef` могут приводиться к `NSString *`, давая вам доступ к объекту `NSString`, содержащему название записи. Например, `NSString *name = (NSString *) ABRecordCopyCompositeName(record);`.

Функции адресной книги верхнего уровня

Прежде чем вы сможете считывать или записывать в адресную книгу, вы должны инициализировать ее. Чтобы получить идентификатор (handle) адресной книги, воспользуйтесь функцией `ABAddressBookCreate`:

```
#import <AddressBook/AddressBook.h>
ABAddressBookRef ab = ABAddressBookCreate();
```

Вы должны явно сохранить любые изменения, сделанные вами в адресной книге. Для этого, когда вы завершите редактирование записей и захотите зафиксировать сделанные изменения, вызовите функцию `ABAddressBookSave`:

```
CFErrorRef error;
BOOL success = ABAddressBookSave(ab, &error);
```

Если вы не уверены в том, надо ли сохранять адресную книгу, воспользуйтесь функцией `ABAddressBookHasUnsavedChanges`:

```
BOOL hasUnsavedChanges = ABAddressBookHasUnsavedChanges(ab);
```

Чтобы добавить или удалить записи, воспользуйтесь функциями `ABAddressBookAddRecord` и `ABAddressBookRemoveRecord`:

```
CFErrorRef error;
BOOL success = ABAddressBookAddRecord(ab, myRecord, &error);
BOOL success = ABAddressBookRemoveRecord(ab, myRecord, &error);
```

Выполнение запросов к адресной книге

Библиотека Address Book предоставляет только базовую функциональность запросов. В ней имеются функции для выполнения запросов на несколько

записей по имени или отдельных записей по индивидуальным ID. Чтобы получить общее количество записей в адресной книге, воспользуйтесь функцией `ABAddressBookGetPersonCount`. Эта функция возвращает `CFIndex`, который приводится к 32-битному целому числу:

```
CFIndex count = ABAddressBookGetPersonCount(ab);
NSLog(@"%ld total entries in the address book", count);
```

Существуют две функции для запроса нескольких записей адресной книги. Обе эти функции возвращают `CFArrayRef`, который может быть приведен к `NSArray *`, т. е. указателю на объект `NSArray`. Чтобы получить список всех контактов в базе данных, воспользуйтесь функцией `ABAddressBookCopyArrayOfAllPeople`:

```
NSArray *array = (NSArray *)ABAddressBookCopyArrayOfAllPeople(ab);
NSLog(@"Retrieved %d contacts\n", [ array count ]);
```

Чтобы в списке контактов найти определенное имя, воспользуйтесь функцией `ABAddressBookCopyPeopleWithName`. Поиск можно производить по имени, фамилии или по тому и другому. Все найденные записи будут возвращены в массиве:

```
NSArray *array = (NSArray *)ABAddressBookCopyPeopleWithName(ab,
    CFSTR("John Appleseed));
```

Как следует из названия функции, возвращаемые объекты являются не действительными объектами адресной книги, а их копиями. Чтобы получить доступ к отдельным записям внутри массива, воспользуйтесь методом `objectAtIndex` класса `NSArray`:

```
ABRecordRef record = [ people objectAtIndex: 0 ];
```

Если вы знаете ID записи контакта, который вы хотите загрузить, то помимо выполнения запросов нескольких контактов вы можете загрузить его напрямую с помощью функции `ABAddressBookGetPersonWithRecordID`:

```
ABRecordRef record = ABAddressBookGetPersonWithRecordID(ab, recordId);
```

Создание записей

Чтобы создать новый контакт, воспользуйтесь функцией `ABPersonCreate`. В результате вы получите пустую запись, в которую потом сможете добавлять информацию:

```
ABRecordRef record = ABPersonCreate();
```

Работа с записями

Имея под рукой `ABRecordRef`, вы можете определить, представляет ли данная запись человека или группу, а в результате сможете получить доступ к более детальной информации. Имя, фамилия и другие составляющие информации — все они являются *свойствами* (properties). Сущность имеет свойства, если она является записью `ABPerson`, а для каждой записи имеется различное количество информации.

Чтобы запросить информацию по заданной записи, воспользуйтесь функцией `ABRecordCopyValue`. Вот прототип этой функции:

```
CFTypeRef ABRecordCopyValue(ABRecordRef record, ABPropertyID property);
```

При своем вызове функция `ABRecordCopyValue` копирует указанное вами свойство и возвращает ссылку:

```
CFStringRef firstName = ABRecordCopyValue(myRecord,
  kABPersonFirstNameProperty);
```

Поскольку свойство `kABPersonFirstNameProperty` является `CFStringRef`, то вы можете привести его к `NSString *`:

```
NSString *firstName = (NSString *) ABRecordCopyValue(myRecord,
  kABPersonFirstNameProperty);
```

Как `CFStringRef` может быть приведено к `NSString *`, так и любое свойство, возвращающее `CFDateRef`, может быть приведено к `NSDate *`:

```
NSDate *birthday = (NSDate *) ABRecordCopyValue(myRecord,
  kABPersonBirthdayProperty);
```

Указанный вами `ABPropertyID` является значением, соответствующим искомой вами информации в записи. В табл. 9.1 приведены свойства для объекта `ABPerson`. Поскольку тип данных, возвращаемый функцией `ABRecordCopyValue`, является общим `CFTypeRef`, то получаемое в результате значение может быть приведено к типу данных, специально созданному для данного свойства.

Таблица 9.1

Свойство	**Описание**	**Тип данных**
`kABPersonFirstNameProperty`	Имя (First Name)	`CFStringRef`
`kABPersonLastNameProperty`	Фамилия (Last Name)	`CFStringRef`

Таблица 9.1 *(окончание)*

Свойство	Описание	Тип данных
kABPersonMiddleNameProperty	Отчество (Middle Name)	CFStringRef
kABPersonPrefixProperty	Обращение (Surname); "Sir"	CFStringRef
kABPersonSuffixProperty	Приставка (Suffix), "Jr." or "Sr."	CFStringRef
kABPersonNicknameProperty	Кличка (Nickname)	CFStringRef
kABPersonFirstNamePhoneticProperty	Имя (First name); фонетическое (phonetic)	CFStringRef
kABPersonLastNamePhoneticPropert	Фамилия (Last name); фонетическая (phonetic)	CFStringRef
kABPersonMiddleNamePhoneticProperty	Отчество (Mid name); фонетическое (phonetic)	CFStringRef
kABPersonOrganizationProperty	Название компании (Company name)	CFStringRef
kABPersonJobTitleProperty	Должность (Job title)	CFStringRef
kABPersonDepartmentProperty	Отдел (Department)	CFStringRef
kABPersonBirthdayProperty	Дата рождения (Birthday)	CFDateRef
kABPersonNoteProperty	Заметки (Notes)	CFStringRef
kABPersonCreationDateProperty	Дата создания (Creation date)	CFDateRef
kABPersonModificationDateProperty	Дата последнего изменения (Date last modified)	CFDateRef
kABPersonKindProperty	Тип контакта (Contact type)	Enumeration

Запись свойств

Чтобы записать какое-либо свойство в запись адресной книги, воспользуйтесь функцией `ABRecordSetValue`:

```
CFErrorRef error;
CFStringRef nickname = CFSTR("Sparky");
BOOL success = ABRecordSetValue(record, kABPersonNicknameProperty,
    nickname, &error);
```

Чтобы удалить какое-либо свойство, воспользуйтесь функцией `ABRecordRemoveValue`:

```
CFErrorRef error;
BOOL success = ABRecordRemoveValue(myRecord, kABPersonNicknameProperty,
                                   &error);
```

Когда вы закончите редактирование записи, не забудьте сохранить адресную книгу:

```
CFErrorRef error;
BOOL success = ABAddressBookSave(ab, &error);
```

Многозначные свойства

Помимо перечисленных выше свойств некоторые записи могут содержать несколько значений. Многозначные свойства (multivalue properties) обрабатываются с помощью механизма индексирования, в котором сначала запрашивается общее количество значений, а затем осуществляется вызов для получения значения с определенным индексом. Сначала получается указатель на многозначные данные с помощью ранее описанного метода `ABRecordCopyValue` и приводится к `ABMultiValueRef`:

```
ABMultiValueRef phoneNumbers = ABRecordCopyValue(myRecord,
  kABPersonPhoneProperty);
```

Затем вы можете использовать ссылку на определение количества значений и получение отдельных значений по индексу. Функция `ABMultiValueGetCount` возвращает количество элементов, а функция `ABMultiValueCopyValueAtIndex` копирует элемент с указанным вами индексом:

```
NSMutableArray *phoneNumbersList = [ [ NSMutableArray alloc ] init ];

CFIndex nPhoneNumbers = ABMultiValueGetCount(phoneNumbers);
```

```
for(int i=0;i<nPhoneNumbers;i++) {
    NSString *phoneNumber = (NSString *)ABMultiValueCopyValueAtIndex(
        phoneNumbers, i);
    [ phoneNumbersList addObject: phoneNumber ];
    [ phoneNumber release ];
}
```

Приведенные в табл. 9.2 типы данных описывают отдельную запись внутри перечисленных многозначных свойств.

Таблица 9.2

Свойство	**Описание**	**Тип данных**
`kABPersonEmailProperty`	Адрес электронной почты (Email addresses)	`CFStringRef`
`kABPersonAddressProperty`	Адрес (Address)	`CFDictionary-Ref`
`kABPersonDateProperty`	Связанные даты (Associated dates)	`CFDateRef`
`kABPersonPhoneProperty`	Номера телефонов (Phone numbers)	`CFStringRef`
`kABPersonInstantMessageProperty`	IM IDs	`CFDictionary-Ref`
`kABPersonURLProperty`	Web-адреса (Web-site URLs)	`CFStringRef`
`kABPersonRelatedNamesProperty`	Родственные имена (Related names)	`CFStringRef`

Помимо самих записей внутри многозначных свойств эти записи имеют метки (label). Метка описывает тип возвращаемой записи. Например, метки отдельных номеров телефонов могут указывать на то, является ли этот номер домашним или мобильным. Метки для адресов могут указывать на домашние или рабочие адреса. Чтобы запросить метку для заданной записи, воспользуйтесь функцией `ABMultiValueCopyLabelAtIndex`:

```
CFStringRef label = ABMultiValueCopyLabelAtIndex(phoneNumbers, i);
```

Конкретные свойства имеют заданный набор меток. В прототипах ABPerson.h заданы следующие метки `CFStringRef`:

```
kABPersonDateProperty
        kABPersonAnniversaryLabel
kABPersonPhoneProperty
        kABPersonPhoneMobileLabel
        kABPersonPhoneMainLabel
        kABPersonPhoneHomeFAXLabel
        kABPersonPhoneWorkFAXLabel
        kABPersonPhonePagerLabel
kABPersonInstantMessageProperty
```

К ключу `kABPersonInstantMessageServiceKey` внутри словаря применяются:

```
        kABPersonInstantMessageServiceYahoo
        kABPersonInstantMessageServiceJabber
        kABPersonInstantMessageServiceMSN
        kABPersonInstantMessageServiceICQ
        kABPersonInstantMessageServiceAIM
kABPersonURLProperty
        kABPersonHomePageLabel
kABPersonRelatedNamesProperty
        kABPersonFatherLabel
        kABPersonMotherLabel
        kABPersonParentLabel
        kABPersonBrotherLabel
        kABPersonSisterLabel
        kABPersonChildLabel
        kABPersonFriendLabel
        kABPersonSpouseLabel
        kABPersonPartnerLabel
        kABPersonAssistantLabel
        kABPersonManagerLabel
```

Многие свойства используют общий набор меток для рабочих, домашних и других мест. Вот эти общие метки:

```
kABWorkLabel
kABHomeLabel
kABOtherLabel
```

Запись многозначных записей

Чтобы добавить какое-либо значение к существующему многозначному свойству, вы должны сначала скопировать многозначный словарь из записи. Затем вы работаете с новой копией, добавляя новые пары "значение/метка" с помощью функции `ABMultiValueAddValueAndLabel`. Наконец, воспользуйтесь функцией `ABRecordSetValue` для записи всего словаря обратно в запись адресной книги, полностью заменяя многозначное свойство. В приведенном далее примере к свойству `kABPersonURLProperty` добавляется новый URL:

```
CFErrorRef error;
ABMultiValueRef URLs = ABRecordCopyValue(myRecord, kABPersonURLProperty);
ABMutableMultiValueRef copyOfURLs = ABMultiValueCreateMutableCopy(URLs);
ABMultiValueAddValueAndLabel(copyOfURLs, "http://www.oreilly.com",
    kABPersonHomePageLabel, NULL);
ABRecordSetValue(myRecord, kABPersonURLProperty, copyOfURLs, &error);
```

Чтобы удалить пару "значение/метка", воспользуйтесь функцией `ABMultiValueRemoveValueAndLabelAtIndex`:

```
CFErrorRef error;
ABMultiValueRef URLs = ABRecordCopyValue(myRecord, kABPersonURLProperty);
ABMutableMultiValueRef copyOfURLs = ABMultiValueCreateMutableCopy(URLs);
ABMultiValueRemoveValueAndLabelAtIndex(copyOfURLs, 0);
ABRecordSetValue(myRecord, kABPersonURLProperty, copyOfURLs, &error);
```

Убедитесь в том, что вы сохранили запись, с помощью функции `ABAddressBookSave`:

```
ABAddressBookSave(ab, &error);
```

Работа со словарями

Записи адресной книги используют словари (dictionaries) для описания адресов и учетных записей (accounts) программ обмена мгновенными сообщениями (instant messenger). Эти словари встроены в многозначные записи. Чтобы получить доступ к ним, скопируйте нужное значение и приведите к типу `NSDictionary *`.

Теперь вы сможете получить доступ к словарю с помощью множества предопределенных ключей:

```
ABMultiValueRef addresses = ABRecordCopyValue(record,
  kABPersonAddressProperty);
CFIndex nAddresses = ABMultiValueGetCount(addresses);
NSLog(@"%d addresses\n", nAddresses);

NSDictionary *address = ABMultiValueCopyValueAtIndex(addresses, 0);
for (id key in address)
{
    NSLog(@"key: %@, value: %@", key, [ address objectForKey: key ]);
}
```

Для свойств словарей, сохраненных в адресной книге, используются следующие ключи:

```
kABPersonAddressProperty
       kABPersonAddressStreetKey
       kABPersonAddressCityKey
       kABPersonAddressStateKey
       kABPersonAddressZIPKey
       kABPersonAddressCountryKey
       kABPersonAddressCountryCodeKey
kABPersonInstantMessengerProperty
       kABPersonInstantMessageServiceKey
       kABPersonInstantMessageUsernameKey
```

Данные изображения

Некоторым контактам сопоставляется изображение. Чтобы получить данные какого-либо изображения, возвращаемые как `CFDataRef`, воспользуйтесь функцией `ABPersonCopyImageData`. Вы можете привести их к типу `NSData *`, который может инициализировать объект `UIImage`. Вы уже изучали класс `UIImage` в предыдущих главах и узнаете о нем еще больше в *главе 10*.

```
if (ABPersonHasImageData(myRecord)) {
    UIImage *addressBookImage = [ UIImage imageWithData:
        (NSData *) ABPersonCopyImageData(myRecord)
    ];
}
```

Для дальнейшего изучения

Кладезем информации о свойствах и функциях адресной книги являются прототипы, находящиеся в каталоге AddressBook.framework. Вы найдете их в папке /Developer/Platforms/iPhoneOS.platform.

Address Book UI

Библиотека Address Book UI предоставляет два ключевых пользовательских интерфейса: контроллер навигации для выбора контактов (people picker) и контроллер представлений для отображения единичного контакта.

Представления для отображения контактов

`ABPersonViewController` предоставляет простой интерфейс для отображения единичного контакта пользователю. Представление для отображения контактов (person view) требует `CFRecordRef`, с которым вы познакомились ранее в этой главе. Эта запись передается представлению для отображения контактов, которое выполняет всю рутинную работу по загрузке и отображению контактной информации человека. Создайте `ABPersonViewController` точно так же, как и любой другой контроллер представлений:

```
ABPersonViewController *viewController = [[ABPersonViewController alloc]
    init
];
```

Назначьте требуемую для отображения запись свойству `displayedPerson`:

```
viewController.displayedPerson = myRecord;
```

Далее, создайте массив свойств, которые вы хотите отобразить пользователю. Отображены будут только те свойства, которые вы укажете, если только контакт не редактируется, в этом случае отображены будут все свойства. Доступные свойства — это те же самые перечислимые значения, о которых вы узнали ранее в этой главе. Каждое из них добавляется как объект `NSNumber`:

```
NSMutableArray *properties = [ [ NSMutableArray alloc ] init ];
[ properties addObject: [ NSNumber numberWithInt:
  kABPersonFirstNameProperty ]];
[ properties addObject: [ NSNumber numberWithInt:
  kABPersonLastNameProperty ]];
```

```
[ properties addObject: [ NSNumber numberWithInt:
  kABPersonOrganizationProperty ]];
viewController.displayedProperties = properties;
```

Если вы захотите, чтобы пользователь мог редактировать контакт, то задайте `YES` в качестве значения свойства `allowsEditing`:

```
viewController.allowsEditing = YES;
```

Выборщики контактов

Если ваше приложение должно отображать список контактов пользователю, то вам поможет в этом класс `ABPeoplePickerNavigationController`. Этот контроллер навигации отображает список контактов и позволяет пользователю выделить в нем один из контактов. После выделения вы можете либо отобразить данный контакт пользователю, либо использовать метод-делегат для задания собственного поведения.

Чтобы создать выборщика контактов (people picker), породите объект `ABPeoplePickerNavigationController`:

```
ABPeoplePickerNavigationController *peoplePicker = [
    [ ABPeoplePickerNavigationController alloc ] init
];
```

Если ваш выборщик будет позволять пользователю просматривать контакт человека, то определите множество свойств, которые может видеть пользователь. По умолчанию пользователю отображаются все имеющиеся поля. Доступные свойства — это те же самые перечислимые значения, о которых вы узнали ранее в этой главе. Каждое из них добавляется как объект `NSNumber`:

```
NSMutableArray *properties = [ [ NSMutableArray alloc ] init ];
[ properties addObject: [ NSNumber numberWithInt:
  kABPersonFirstNameProperty ]];
[ properties addObject: [ NSNumber numberWithInt:
  kABPersonLastNameProperty ]];
[ properties addObject: [ NSNumber numberWithInt:
  kABPersonOrganizationProperty ]];
peoplePicker.displayedProperties = properties;
```

Чтобы назначить делегата для получения уведомлений, когда пользователь выделит тот или иной контакт, назначьте вашего делегата свойству `peoplePickerDelegate` выборщика:

```
peoplePicker.peoplePickerDelegate = self;
```

Затем вы сможете добавить этот контроллер навигации к окну или существующему представлению:

```
[ window addSubview: [ peoplePicker view ] ];
```

Методы-делегаты

Для различных действий выборщика контактов вызываются три метода-делегата. Если пользователь решит отменить выделение, то уведомляется `peoplePickerNavigationControllerDidCancel`. Этот метод должен корректно обработать отказ в выделении контакта:

```
- (void)peoplePickerNavigationControllerDidCancel:
    (ABPeoplePickerNavigationController *)peoplePicker
{
  /* Код для обработки отмены выделения */
}
```

Когда выделяется контакт, делегат получает уведомление вместе с `ABRecordRef` выделенного контакта и запрос на необходимость продолжения. Если делегат возвращает `YES`, то выделенный контакт отображается пользователю вместе с внутренним `ABPersonViewController`, созданным выборщиком контактов. Если ваше приложение будет выполнять какое-либо другое действие, то верните `NO` и перейдите к другому экрану:

```
- (BOOL)peoplePickerNavigationController:
    (ABPeoplePickerNavigationController *)peoplePicker
    shouldContinueAfterSelectingPerson:(ABRecordRef)person
{
    /* Код для обработки выделения контакта */
}
```

Если пользователь коснется свойств в отображенном контакте, то делегату отправится третье уведомление, содержащее выбранное свойство. Снова будет получен запрос на необходимость продолжения. Если ваше приложение

при выделении пользователем свойства будет выполнять какое-либо другое действие, то верните `NO` и перейдите к другому экрану:

```
- (BOOL)peoplePickerNavigationController:
     (ABPeoplePickerNavigationController *)peoplePicker
    shouldContinueAfterSelectingPerson:(ABRecordRef)person
    property:(ABPropertyID)property
    identifier:(ABMultiValueIdentifier)identifier
{
    /* Код для обработки выделения свойства */
}
```

Для дальнейшего изучения

Дополнительную информацию о контроллере адресной книги и свойствах можно найти в каталогах AddressBookUI.framework. Вы найдете их в заголовках вашего SDK в папке /Developer/Platforms/iPhoneOS.platform.

Глава 10

Проектирование UI Kit для опытных

В *главе 3* вы познакомились с библиотекой UI Kit, являющейся сердцем всех приложений GUI на iPhone. Данная глава описывает более сложные и эстетически богатые компоненты библиотеки пользовательского интерфейса iPhone и показывает вам, как сделать ваше собственное программное обеспечение таким же эффектным, как и поставляемые Apple приложения.

В данной главе будут освещены следующие расширенные компоненты UI Kit.

- *Общие элементы управления.* UI Kit предоставляет набор элементов управления (controls), в том числе переключатели, кнопки, сегментированные элементы управления, полосы прокрутки и др. Элементы управления используются на панелях навигации, в ячейках таблиц и в других визуальных элементах. Класс `UIControl` является базовым классом для элементов управления, предоставляющим стандартизованный механизм уведомлений о событиях.
- *Таблицы настроек и списки разделов.* Класс `UITableView`, с которым вы познакомились в *главе 3*, поддерживает альтернативные раскладки для отображения таблиц настроек (preferences tables) и списков разделов (section lists). Таблицы настроек используются для управления сгруппированными настройками программ или отображения структурированной информации. Списки разделов — это списки контактов, музыки или других индексированных данных, представленные в стиле старой картотеки (Rolodex).

- *Индикаторы прогресса и активности.* 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. Индикаторы активности (activity indicators) показывают пользователю, что то или иное действие находится в стадии выполнения. Приложение может указать с их помощью, что оно использует сеть, или предоставить индикаторы общего назначения, информирующие пользователя о том, что необходимо подождать.
- *Изображения.* UI Kit предоставляет классы для обработки и отображения изображений. Эти классы могут загружать наиболее распространенные типы изображений, а также отображать, изменять, наслаивать и отсекать их в любом месте экрана. Кроме того, классы изображений могут быть анимированы.
- *Текстовые поля свойств клавиатуры.* iPhone поддерживает несколько различных стилей клавиатуры, которые широко применяются в приложениях для организации пользовательского ввода различного типа. Приложение Mail от Apple использует клавиатуру, содержащую специальные символы для ввода адресов электронной почты, а Safari использует другую клавиатуру, удобную для ввода URL, имеющую кнопку **.COM**. Текстовые поля (text fields) содержат специальные свойства для описания стиля клавиатуры, отображаемой при осуществлении ввода в эти поля.
- *Выборщики данных и представления выборщиков.* Выборщики (pickers) предоставляют унифицированный метод ввода параметров, 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 элементами управления, позволяющими осуществлять выбор собственных дат, времени и временных периодов.
- *Панели вкладок.* Панели вкладок (tab bars) предоставляют ряд кнопок быстрого доступа вдоль нижней части экрана iPhone, позволяя программисту логически сгруппировать основные функции приложения. Кроме того, это весьма распространенный способ разделить представления различных данных. Приложение iPod использует панели вкладок для разделения списков файлов для воспроизведения (playlists), исполнителей, песен и видеоклипов друг от друга, а приложение телефона использует

панель кнопок для быстрого доступа к различным функциям телефона, таким как клавиатура или список контактов.

- *Сенсоры и информация об устройстве.* iPhone содержит множество сенсоров, например, акселерометр (accelerometer) и датчик приближения (proximity sensor). Остальные сенсоры, как, например, датчик ориентации, используются для автоматического управления поворотами экрана. Класс `UIDevice` позволяет также считывать определенную информацию уровня операционной системы, например, модель устройства, версию программного обеспечения и уникальный идентификатор. В этом разделе вы узнаете, как считывать и понимать различные индикаторы, имеющиеся в iPhone.
- *Представления прокруток.* Многие объекты с большим количеством содержимого для отображения используют представления прокруток (scroll views). Представления прокруток позволяют прокручивать содержимое, чтобы оно попало в поле зрения, и увеличивать его масштаб. Вы косвенно использовали их в таблицах и списках, а здесь вы увидите, как они работают.
- *Web-представления.* Класс Web-представления (web view) является встроенным в библиотеку UI Kit, позволяя приложениям отображать Web-страницы. Это очень мощное средство для сетевых инструментов, которые могут выбрать использование Web-страниц для обновления окна "последних новостей" или отображения другой информации. Web-представления могут также отображать небольшие PDF-файлы и изображения.

Элементы управления

Элементы управления (controls) — это разнообразные служебные классы, созданные для расширения возможностей пользовательского интерфейса приложений. UI Kit поддерживает множество различных типов элементов управления, порожденных от стандартного класса `UIControl`. В SDK присутствуют следующие элементы управления: переключатели (switches), полосы прокрутки (sliders), сегментированные элементы управления (segmented controls), кнопки (buttons), элементы управления страницей (page controls) и текстовые поля (text fields). Элементы управления — это расширения классов, порожденных от `UIView`, которые могут быть прикреплены непосредственно к панелям навигации, ячейкам таблицы и даже более крупным объектам.

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 широко распространенного элемента управления настольных систем был создан отдельный элемент управления для iPhone.

Базовый класс *UIControl*

Элементы управления порождены от базового класса `UIControl`, который предоставляет унифицированный интерфейс для уведомления о событиях, изменения состояний и отображения свойств. Класс `UIControl` порожден от класса `UIView`, поэтому каждый элемент управления обладает многими свойствами представления, включая способность прикрепляться к другим представлениям. Все элементы управления используют общее множество свойств и методов.

Свойства

- `enabled` — элементы управления разрешены по умолчанию. Чтобы запретить какой-либо элемент управления, установите свойство `enabled` в `NO`, в результате чего элемент управления будет игнорировать все события касаний. Запрещенный элемент управления будет визуализироваться иначе, т. е. будет отображаться в сером цвете. Хотя сама визуализация отдана на откуп подклассу элемента управления, свойство для этого существует в `UIControl`.
- `selected` — когда пользователь выделяет какой-либо элемент управления, класс `UIControl` устанавливает свойство `selected` в `YES`. Подкласс иногда использует этот факт для того, чтобы элемент управления самовыделялся или вел себя иначе.
- `contentVerticalAlignment` определяет, как должно располагаться содержимое элемента управления по вертикали. По умолчанию содержимое выравнивается по верхнему краю, но может потребоваться изменить это, к примеру, для текстовых полей на `UIControlContentVerticalAlignmentCenter`. Возможные значения для этого свойства:
  - `UIControlContentVerticalAlignmentCenter`;
  - `UIControlContentVerticalAlignmentTop`;

- `UIControlContentVerticalAlignmentBottom`;
- `UIControlContentVerticalAlignmentFill`.

- `contentHorizontalAlignment` определяет, как должно располагаться содержимое текстового поля по горизонтали. Возможные значения для этого свойства:
  - `UIControlContentHorizontalAlignmentCenter`;
  - `UIControlContentHorizontalAlignmentLeft`;
  - `UIControlContentHorizontalAlignmentRight`;
  - `UIControlContentHorizontalAlignmentFill`.

Уведомления о событиях

Класс `UIControl` предоставляет стандартный механизм для регистрации и получения событий. Вы можете указывать вашему элементу управления уведомлять метод в вашем классе-делегате всякий раз, когда происходит определенное событие. Чтобы зарегистрировать событие, воспользуйтесь методом `addTarget`:

```
[ myControl addTarget: myDelegate action: @selector(myActionMethod:)
    forControlEvents: UIControlEventValueChanged
];
```

События могут быть объединены посредством логической операции `OR`, позволяя тем самым указывать несколько событий за один вызов `addTarget`. Приведенные далее события поддерживаются классом `UIControl` и поэтому применяются ко всем элементам управления, если не указано иное.

- `UIControlEventTouchDown` уведомляет обо всех отдельных событиях касаний (touch down). Это происходит, когда пользователь касается экрана или на экран опускаются еще один или несколько пальцев;
- `UIControlEventTouchDownRepeat` уведомляет обо всех событиях множественных касаний, когда количество касаний превышает 1. Это происходит, когда пользователь опускает на экран второй, третий или четвертый палец;
- `UIControlEventTouchDragInside` уведомляет, когда касание перетаскивается в пределах окна элемента управления;
- `UIControlEventTouchDragOutside` уведомляет, когда касание перетаскивается вне пределов окна элемента управления;

- `UIControlEventTouchDragEnter` уведомляет, когда касание перетаскивается извне внутрь окна элемента управления;
- `UIControlEventTouchDragExit` уведомляет, когда коснувшийся палец перемещается из окна элемента управления вовне;
- `UIControlEventTouchUpInside` уведомляет обо всех событиях поднятия пальца (touch up) внутри элемента управления;
- `UIControlEventTouchUpOutside` уведомляет обо всех событиях поднятия пальца (touch up), произошедших вне элемента управления. Уведомления не будут отправляться, если касание было порождено внутри элемента управления;
- `UIControlEventTouchCancel` уведомляет обо всех событиях отмены касания, когда касание было отменено путем помещения на экран слишком большого количества пальцев или было прервано блокировкой или телефонным звонком;
- `UIControlEventValueChanged` уведомляет, когда меняется значение элемента управления. Используется в бегунках, сегментированных элементах управления и других элементах управления, использующих значения. Вы можете настроить элементы управления бегунками для уведомления о том, что бегунок был отпущен, или о том, что он перетаскивается;
- `UIControlEventEditingDidBegin` уведомляет о том, что внутри текстового элемента управления началось редактирование;
- `UIControlEventEditingChanged` уведомляет о том, что внутри текстового элемента управления был изменен текст;
- `UIControlEventEditingDidEnd` уведомляет о том, что внутри текстового элемента управления завершилось редактирование;
- `UIControlEventEditingDidEndOnExit` уведомляет о том, что внутри текстового элемента управления завершилось редактирование путем нажатия клавиши возврата (или ее эквивалента);
- `UIControlEventAllTouchEvents` уведомляет обо всех событиях касаний;
- `UIControlEventAllEditingEvents` уведомляет обо всех событиях редактирования текста;
- `UIControlEventAllEvents` уведомляет обо всех событиях.

Помимо заданных по умолчанию событий собственные классы элементов управления для описания своих событий могут использовать диапазон значений от 0x0F000000 до 0x0FFFFFFF.

Чтобы удалить какое-либо действие для одного или нескольких событий, воспользуйтесь методом `removeTarget` класса `UIControl`. Значение `nil` удалит все действия для заданного события:

```
[ myControl removeTarget: myDelegate action: nil
    forControlEvents: UIControlEventAllEvents
];
```

Чтобы получить список всех действий, заданных для элемента управления, воспользуйтесь методом `allTargets`. Он возвращает `NSSet`, содержащий полный список всех событий:

```
NSSet *myActions = [ myControl allTargets ];
```

Иначе вы можете получить список всех действий для заданного события с помощью метода `actionsForTarget`:

```
NSArray *myActions = [ myControl actionsForTarget:
                        UIControlEventValueChanged ];
```

Если вы создаете собственный класс элемента управления, то можете отправить уведомления о базовых событиях `UIControl` или собственных событиях с помощью метода `sendActionsForControlEvents`. Например, если меняется значение вашего элемента управления, то вы можете послать уведомление о соответствующем событии, которое будет передано всем получателям уведомлений о событии, указанным в коде, использующем ваш элемент управления. Вот пример этого:

```
[ self sendActionsForControlEvents: UIControlEventValueChanged ];
```

Когда о событии уведомляется класс-делегат, то он получает указатель на отправителя события. Ваш метод действия должен соответствовать конструкции, приведенной в примере далее, который обрабатывает данное событие для сегментированного элемента управления:

```
- (void) myAction: (id)sender
{
    UISegmentedControl *control = (UISegmentedControl *) sender;
    if (control == myControl1) {
        /* Запрашиваем значение у элемента управления */

        /* Отвечаем на действие для myControl1 */
    }
}
```

Сегментированные элементы управления

Сегментированные элементы управления (segmented controls) заменяют переключатели, применяемые в настольных операционных системах, а вместо них предоставляют интерфейс, схожий с тем, который имеется на передней панели посудомоечной машины или микроволновой печи. Пользователь видит панель инструментов, на которой нажатие одной кнопки приводит к поднятию всех остальных. Сегментированные элементы управления полезны при ограниченном количестве связанных вариантов выбора, доступных для одного элемента.

Создание элемента управления

Как и в случае с другими классами представлений, сегментированный элемент управления инициализируется с фреймом. Координатами фрейма являются смещение относительно представления, в котором размещается данный элемент управления, являющийся обычно таблицей настроек или панелью навигации:

```
UISegmentedControl *segmentedControl = [ [ UISegmentedControl alloc ]
     initWithItems: nil
];

segmentedControl.segmentedControlStyle = UISegmentedControlStyleBar;
```

В зависимости от того, где используются элементы управления, можно выбрать один из трех различных стилей сегментированного элемента управления (табл. 10.1).

Таблица 10.1

Стиль	Описание
`UISegmentedControlStylePlain`	Большие белые кнопки с серой границей, подходит для ячеек настроек
`UISegmentedControlStyleBordered`	Большие белые кнопки с черной границей, подходит для ячеек таблицы
`UISegmentedControlStyleBar`	Маленькие кнопки, подходит для панелей навигации

Если вы используете стиль `UISegmentedControlStyleBar`, то с помощью свойства `tintColor` элемента управления можете также задать оттенок всего элемента управления:

```
UIColor *myTint = [ [ UIColor alloc ] initWithRed: 0.75
    green: 1.0
    blue: 0.75
    alpha: 1.0
];

segmentedControl.tintColor = myTint;
```

Добавление сегментов

Каждый сегмент внутри сегментированного элемента представления выражен кнопкой, содержащей метку или изображение. Для каждого возможного варианта выбора в вашем элементе управления должен быть создан отдельный сегмент. Сегментов может быть столько, сколько поместится на экране, но в один момент времени пользователь может выбрать только один сегмент. Пример PageDemo из *главы 3* иллюстрирует создание кнопок для кроликов и пони:

```
[ segmentedControl insertSegmentWithTitle: @"Bunnies"
    atIndex: 0 animated: NO
];

[ segmentedControl insertSegmentWithTitle: @"Ponies"
    atIndex: 1 animated: NO
];
```

Каждой кнопке сопоставлен порядковый номер (0, 1, 2 и т. д.). Числовое значение предназначено для упорядочивания кнопок и служит в вашем приложении идентификатором кнопки. Чтобы добавить сегмент, содержащий изображение, воспользуйтесь методом `insertSegmentWithImage`:

```
UIImage *myBunnies = [ UIImage applicationImageNamed: @"bunnies.png" ];

[ segmentedControl insertSegmentWithImage: myBunnies
    atIndex: 0 animated: NO
];
```

Сегменты могут быть удалены. Чтобы удалить какой-либо отдельный сегмент, воспользуйтесь методом `removeSegment`:

```
[ segmentedControl removeSegmentAtIndex: 1 animated: YES ];
```

Чтобы удалить все сегменты за раз, вызовите метод `removeAllSegments`. Это приведет к тому, что элемент управления скроет свои кнопки:

```
[ segmentedControl removeAllSegments ];
```

Заголовки сегментов

Если когда-либо понадобится изменить заголовок какой-либо кнопки, воспользуйтесь методом `setTitle`:

```
[ segmentedControl setTitle:@"Unicorms" forSegment: 0 ];
```

Кроме того, вы можете считать заголовки с помощью метода `titleForSegmentAtIndex`:

```
NSString *myTitle = [ segmentedControl titleForSegmentAtIndex: 0 ];
```

Изображения

Помимо текста сегментированные элементы управления могут отображать на кнопках изображения. Любые используемые изображения должны храниться в программной папке приложения. Для этого их необходимо перетащить в папку Resources проекта Xcode. Изображение может быть добавлено к существующему сегменту с помощью метода `setImage`. Этот метод схож с методом `insertSegmentWithImage`, с которым вы уже познакомились, но работает с существующим сегментом:

```
[ segmentedControl setImage: [ UIImage
  applicationImageNamed:@"unicorns.png" ]
    forSegmentAtIndex: 0
];
```

Также вы можете считать изображение сегмента с помощью метода `imageForSegmentAtIndex`:

```
UIImage *myImage = [ segmentedControl imageForSegmentAtIndex: 0 ];
```

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 слиш-

ком велико. В связи с этим нужно аккуратно разрабатывать изображения для кнопок, чтобы они помещались на кнопке. Вы можете вручную установить ширину сегмента с помощью метода `setWidth`:

```
[ segmentedControl setWidth: 64.0 forSegmentAtIndex: 0 ];
```

Мгновенные щелчки

По умолчанию сегментированный элемент управления позволяет пользователю удерживать нажатой кнопку до тех пор, пока не будет выбрана другая. Такое поведение можно немного изменить на автоматическое отпускание кнопки сразу же после ее нажатия. Для реализации данного поведения установите свойство `momentary` в `YES`:

```
segmentedControl.momentary = YES;
```

Инициализация сегмента по умолчанию

По умолчанию не выделяется ни один сегмент, если вы не указали иное. Чтобы задать сегмент по умолчанию, задайте свойство `selectedSegmentIndex`:

```
segmentedControl.selectedSegmentIndex = 0;
```

Отображение элемента управления

После того как элемент управления будет настроен, его можно отобразить, добавив как подчиненное представление к объекту любого типа, который сможет его принять, или прикрепив его как представление к панели навигации или другому объекту. Далее приведены соответствующие примеры.

Добавление к родительскому представлению:

```
[ parentView addSubview: segmentedControl ];
```

Добавление к панели навигации (посредством контроллера представлений):

```
self.navigationItem.titleView = segmentedControl;
```

Считывание элемента управления

Чтобы прочитать текущее значение сегментированного элемента управления, считайте свойство `selectedSegmentIndex`. Это свойство предоставляет значение, соответствующее номеру сегмента, выделенному на текущий момент.

Данное значение задается на основе числа, присвоенного сегменту, когда он впервые был вставлен в элемент управления:

```
int x = segmentedControl.selectedSegmentIndex;
```

Чтобы получать уведомления при нажатии кнопки, добавьте действие для события `UIControlEventValueChanged`, воспользовавшись методом `addTarget` класса `UIControl`. Затем ваш класс-получатель уведомления должен считать порядковый номер выделенного элемента управления:

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

Ваш класс действия будет вызываться всякий раз, когда будет выделяться новый сегмент:

```
- (void) controlPressed: (id)sender
{
    UISegmentedControl *control = (UISegmentedControl *) sender;
    if (control == mySegmentedControl) {
      int x = control.selectedSegmentIndex;

      /* Код для действий в ответ на изменение сегмента */
    }
}
```

Переключатели

Так же как сегментированный элемент управления заменяет переключатель, переключатели (switches) заменяют флажки (checkboxes). Элементы управления этого типа используются для включения или выключения каких-либо функциональных возможностей. Переключатели значительно проще в использовании, но, тем не менее, могут быть настроены до определенной степени.

Создание элемента управления

Переключающий элемент управления инициализируется с помощью стандартного метода `initWithFrame`, однако передаваемый в него размер фрейма игнорируется. Вместо этого переключатель определяет собственный наибо-

лее подходящий размер. Данный метод позволяет вам определить только начало координат относительно класса, к которому он будет прикреплен, например, ячейки таблицы или родительского представления. Вы можете задать в качестве размера `0x0`, понимая, что он будет автоматически заменен:

```
UISwitch *switch = [ [ UISwitch alloc ]
    initWithFrame: CGRectMake(170.0, 5.0, 0.0, 0.0)
];
```

Альтернативные цвета

Хотя это и не поддерживается в SDK, тем не менее, вы можете указать переключателю использовать при активации вместо стандартного синего цвета ярко-оранжевый предупреждающий цвет. Функции, которые могут повлиять на быстродействие системы или же послужить причиной других серьезных последствий, могут использовать такие предупреждающие цвета с помощью скрытого метода `setAlternateColors`. Это особенно полезно при распространении приложений для отладки или рецензирования, обозначая, таким образом, элементы управления, которые будут удалены при выпуске приложения:

```
[ switch setAlternateColors: YES ];
```

> ***Внимание!***
>
> Недокументированный API является предметом постоянных изменений. Кроме того, если вы используете недокументированный API, то вашему приложению может быть отказано в публикации в магазине iTunes, поэтому лучше удалите эту функциональность до выхода в свет вашего приложения.

Отображение элемента управления

После инициализации элемента управления его можно отобразить, добавив его как подчиненное представление к объекту любого типа, который может его разместить, или же прикрепив его как представление к навигационному заголовку или другому объекту. Далее приведены соответствующие примеры.

Добавление к родительскому представлению:

```
[ parentView addSubview: switch ];
```

Добавление к панели навигации (посредством контроллера представлений):

```
self.navigationItem.titleView = switch;
```

Расположение переключателя

Значение переключателя может быть считано посредством его свойства `on`. Оно предоставляет булево значение, указывающее на то, был ли активирован переключатель:

```
BOOL switchPosition = switch.on;
```

Также вы можете активировать переключатель в вашем коде с помощью метода `setOn`. Вот пример:

```
[ switch setOn: YES animated: YES ];
```

Чтобы получать уведомления при изменении состояния переключателя, добавьте действие для события `UIControlEventValueChanged`, воспользовавшись методом `addTarget` класса `UIControl`. Затем ваш класс действия считает значение переключателя:

```
[ switch addTarget: self action: @selector(switchStatusChanged:)
    forControlEvents:UIControlEventValueChanged
];
```

Ваш класс действия будет вызываться всякий раз, когда состояние переключателя будет меняться:

```
- (void) switchStatusChanged: (id)sender
{
    UISwitch *control = (UISwitch *) sender;
    if (control == mySwitch) {
      BOOL on = control.on;

      /* Код для действий в ответ на изменение состояния переключателя */
    }
}
```

Полосы прокрутки

Полосы прокрутки (sliders) предоставляют видимый диапазон, который пользователь может менять с помощью бегунка, и настраиваются, чтобы вместить диапазон значений. Вы можете задать диапазоны значений для бегунка, добавить на его концы изображения, а также сделать другие настройки внешнего вида. Бегунок идеально подходит для представления вариантов выбора с

широким диапазоном численных значений, например, настройка уровня громкости, элементы управления чувствительностью и т. п. Apple просто обязана достаточно хорошо определить полосы прокрутки, чтобы перенести их в iPhone, поскольку они широко распространены в настольных системах. Версия бегунков для iPhone более приспособлена для управления пальцами.

Создание элемента управления

Элемент управления "полоса прокрутки" является стандартным объектом `UIControl`. Вы можете инициализировать его, вызвав метод элемента управления `initWithFrame`, чтобы задать начало координат и его ширину. Высота фрейма игнорируется, а поэтому может быть задана как `0x0`:

```
UISlider *slider = [ [ UISlider alloc ] initWithFrame:
    CGRectMake(0.0, 0.0, 200.0, 0.0)
];
```

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. Для определения этих значений класс `UISlider` предоставляет два свойства — `minimumValue` и `maximumValue`:

```
slider minimumValue: 0.0;
slider maximumValue: 100.0;
```

Кроме того, теперь вы можете определить значение по умолчанию для бегунка, задав свойство `value`:

```
slider.value = 50.0;
```

Элемент управления полосой прокрутки может отображать изображения на любом своем конце. Это можно задать таким же способом, как и в сегментированном элементе управления. Скопируйте изображения в папку Resources в Xcode. В результате при установке на iPhone они будут скопированы в программную папку приложения. Использование изображений приводит к увеличению длины полосы прокрутки, поэтому убедитесь в том, что вы увеличили ширину элемента управления в соответствии с размерами изображений:

```
[ slider setMinimumTrackImage:
    [ UIImage applicationImageNamed:@"min.png" ]
    forState: UIControlStateNormal
];
```

```
[ slider setMaximumTrackImage:
    [ UIImage applicationImageNamed:@"max.png" ]
    forState: UIControlStateNormal
];
```

Для различных состояний бегунка можно показывать разные изображения. Существуют следующие состояния:

- *UIControlStateNormal*;
- *UIControlStateHighlighted*;
- *UIControlStateDisabled*;
- *UIControlStateSelected*.

Для отладки существует недокументированный API, позволяющий отображать значения бегунка внутри элемента управления. Для отображения этих значений рядом с бегунком вызовите метод `setShowValue`:

```
[ slider setShowValue: YES ];
```

> ***Внимание!***
>
> Недокументированный API является предметом постоянных изменений. Кроме того, если вы используете недокументированный API, то вашему приложению может быть отказано в публикации в магазине iTunes, поэтому лучше удалите эту функциональность до выхода в свет вашего приложения.

Отображение элемента управления

После инициализации полосы прокрутки ее можно отобразить, добавив как подчиненное представление к объекту любого типа, который может его разместить, либо прикрепив ее как представление к навигационному заголовку или другому объекту. Далее приведены соответствующие примеры.

Добавление к родительскому представлению:

```
[ parentView addSubview: slider ];
```

Добавление к панели навигации (посредством контроллера представлений):

```
self.navigationItem.titleView = slider;
```

Считывание элемента управления

Полоса прокрутки считывает значение с плавающей точкой в пределах диапазона, указанного вами в момент создания данного элемента управления. Само значение можно запросить с помощью свойства `value`:

```
float value = slider.value;
```

Чтобы получать уведомления при изменении значений бегунка, добавьте действие для события `UIControlEventValueChanged`, воспользовавшись методом `addTarget` класса `UIControl`. Затем ваш класс действия считает значение бегунка:

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

Ваш класс действия будет вызываться всякий раз, когда бегунок будет перетаскиваться в новое положение:

```
- (void) sliderValueChanged: (id)sender
{
    UISlider *control = (UISlider *) sender;
    if (control == mySlider) {
      float value = control.value;

      /* Код для действий в ответ на изменение значения бегунка */
    }
}
```

Чтобы данное событие порождалось при каждом перетаскивании бегунка, задайте свойство бегунка `continuous`:

```
slider.continuous = YES;
```

Текстовые поля

В *главе 3* вы познакомились с классом `UITextField`, который может использоваться для добавления текстовых ячеек к таблицам и другим объектам. Класс `UITextField` наследует от `UIControl`, поэтому вы можете использовать многие из свойств класса `UIControl` для настройки поведения объекта `UITextField`.

Параметры стиля

Помимо стилевых параметров, рассмотренных в *главе 3*, элемент управления `UITextField` позволяет задавать выравнивание, стиль границ и целый ряд других параметров внешнего вида. Далее приведены все эти свойства.

- `textAlignment` определяет то, как должен располагаться текст внутри элемента управления. По умолчанию содержимое выравнивается по левому краю:
  - `UITextAlignmentLeft`;
  - `UITextAlignmentRight`;
  - `UITextAlignmentCenter`.
- `borderStyle` задает стиль границы, окружающей текстовый элемент управления. По умолчанию граница отсутствует. Вы можете использовать перечисленные далее значения. Если применяется собственное фоновое изображение, то заданный этим свойством стиль будет игнорироваться:
  - `UITextBorderStyleNone`;
  - `UITextBorderStyleLine`;
  - `UITextBorderStyleBezel`;
  - `UITextBorderStyleRoundedRect`.
- `placeholder` отображает строку в виде серого заполнителя (placeholder) для пустых текстовых полей. Это значение отображается, если текстовое поле еще не было отредактировано и в нем нет значения. Принимает объект `NSString`.
- `clearsOnBeginEditing`. Если текстовое поле должны быть очищено, когда пользователь его коснется, то установите в качестве булева значения этого свойства `YES`. По умолчанию текстовое поле перемещает курсор в то место текстового поля, которого коснулся пользователь, и не удаляет текст.
- `adjustsFontSizeToFitWidth`. В результате установки этого свойства в `YES` размер шрифта текста автоматически уменьшается, чтобы поместиться в текстовом окне. По умолчанию сохраняется исходный размер шрифта, позволяя длинному тексту прокручиваться в представлении.
- `background` принимает объект `UIImage` и устанавливает его в качестве фона текстового поля. В этом случае стиль границы, если он задан, не учитывается.

- `clearButtonMode` определяет поведение кнопки очистки (clear button). Кнопка очистки — это небольшая кнопка ×, отображаемая справа от текстового поля, позволяющая пользователю очищать весь текст. По умолчанию это свойство имеет значение `UITextFieldViewNever`, скрывающее кнопку очистки. Вы можете задать один из следующих режимов:
  - `UITextFieldViewModeNever`;
  - `UITextFieldViewModeWhileEditing`;
  - `UITextFieldViewModeUnlessEditing`;
  - `UITextFieldViewModeAlways`.
- `LeftView`, `leftViewMode`, `rightView`, `rightViewMode`. Эти свойства позволяют вам прикреплять производные от класса `UIView` справа или слева от текстового поля. Чаще всего к текстовым полям прикрепляются объекты `UIButton`, такие как увеличительное стекло (magnifying glasses) или кнопки закладок (bookmarks). Каждое представление имеет соответствующий режим, который может быть задан с помощью тех же самых значений, что и свойство `clearButtonmode`.

Визуализация подмен

Помимо изменения стилевых параметров самого объекта `UITextField` для настройки визуализации текстового поля вы можете переопределить ряд методов в подклассах объекта `UITextField`. Эти методы возвращают структуру `CGRect`, определяющую границы для каждого компонента текстового поля. Если вы создаете пользовательский класс на базе `UITextField`, то можете переопределить эти методы, чтобы изменить одну или несколько границ. Никогда не вызывайте эти методы напрямую; они являются обратными вызовами для среды исполнения iPhone. Далее приведен соответствующий пример:

```
- (CGRect)clearButtonRectForBounds: (CGRect) bounds {
    return CGRectMake(bounds.origin.x + bounds.size.x - 50,
                      bounds.origin.y + bounds.size.y - 20, 16, 16);
}
```

При создании подкласса `UITextField` доступны следующие подмены:

- `borderRectForBounds` задает прямоугольник рамки;
- `textRectForBounds` — границы отображаемого текста;
- `placeholderRectForBounds` — границы текста "заполнителя" (placeholder);

- `editingRectForBounds` — границы текста при редактировании;
- `clearButtonRectForBounds` — границы, в пределах которых отображать кнопку очистки;
- `leftViewRectForBounds` — границы, в пределах которых отображать левое представление;
- `rightViewRectForBounds` — границы, в пределах которых отображать правое представление.

Методы-делегаты

Вы можете назначить делегата для `UITextField` с помощью свойства класса `delegate`. Этот делегат получает целый ряд различных событий, которые могут быть подменены для получения уведомлений об определенных событиях, происходящих в текстовом поле. Каждому методу предоставляется указатель на текстовое поле, в котором произошло данное действие, что позволяет вам обрабатывать несколько текстовых полей в рамках одного делегата. Доступны следующие методы-делегаты.

- `- (BOOL)textFieldShouldBeginEditing:(UITextField *)textField;`

  Возвращает булево значение, указывающее, можно ли разрешить начало редактирования текстового поля. Чтобы вообще запретить редактирование текстового поля, этот метод всегда должен возвращать `NO` (хотя гораздо легче просто запретить элемент управления). Этот метод полезен в случаях, когда собственное текстовое поле требует предпринять какие-либо другие действия до того, как сделать его доступным для чтения и записи.

- `- (void)textFieldDidBeginEditing:(UITextField *)textField;`

  Вызывается всякий раз, когда пользователь касается внутри текстового поля, чтобы приступить к редактированию. Текстовое поле становится первым отвечающим объектом, когда он приступает к редактированию.

- `- (BOOL)textFieldShouldEndEditing:(UITextField *)textField;`

  Возвращает булево значение, указывающее, можно ли разрешить завершение редактирования текстового поля. Когда редактирование завершено, текстовое поле остается первым отвечающим объектом. Чтобы предотвратить исчезновение поля после завершения пользователем редактирования, верните `NO`. Этот метод может оказаться полезным в приложениях обмена сообщениями, в которых текстовое поле всегда должно оставаться активным.

- `- (void)textFieldDidEndEditing:(UITextField *)textField;`

  Вызывается всякий раз, когда завершается редактирование в заданном текстовом поле, а текстовое поле остается в качестве первого отвечающего объекта.

- `- (BOOL)textField:(UITextField *)textField,`
  `shouldChangeCharactersInRange:`
  `(NSRange)range, replacementString:(NSString *)string;`

  Вызывается, когда пользователь вызывает автозамену, чтобы поменять заданный текст на предлагаемый. Это удобно, если вы добавляете в ваше приложение возможность отмены (undo) и хотите отслеживать последние изменения, сделанные в поле, или же вести журнал всех действий по редактированию с целью аудита. Чтобы предотвратить какое-либо изменение текста, верните `NO`. Параметры метода предоставляют объект `NSRange`, указывающий положение изменяемого текста. Также предоставляются предлагаемые замены текста.

- `- (BOOL)textFieldShouldClear:(UITextField *)textField;`

  Возвращает булево значение, указывающее на то, можно ли разрешить очистку текстового поля по запросу пользователя. Это удобно, если до очистки текстового поля должны произойти определенные события, или же если нельзя разрешать очистку текстового поля, пока не выполнены определенные условия.

- `- (BOOL)textFieldShouldReturn:(UITextField *)textField;`

  Возвращает булево значение, указывающее, можно ли завершать редактирование нажатием клавиши возврата. Если вы хотите завершать редактирование, когда пользователь нажимает клавишу возврата, вызовите здесь метод `resignFirstResponder`. В результате будет завершено редактирование текстового поля, а клавиатура будет отменена:

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

  При нажатии пользователем клавиши возврата приложение может предпринимать и другие действия. Например, может начинаться поиск или верификация информации, введенной в текстовое поле.

Уведомления

Поскольку класс `UITextField` порожден от класса `UIControl`, то система уведомлений, используемая в классе `UIControl`, доступна и для текстовых полей. Помимо стандартных событий класса `UIControl` вы можете использовать следующие собственные события, специфичные для класса `UITextField`:

- `UITextFieldTextDidBeginEditingNotification` инициируется, когда текстовое поле переходит в режим редактирования. Свойство уведомления `object` предоставляет текст для редактирования;
- `UITextFieldTextDidChangeNotification` инициируется, когда текст в поле изменяется. Свойство уведомления `object` хранит измененный текст;
- `UITextFieldTextDidEndEditingNotification` инициируется, когда текстовое поле выходит из режима редактирования. Свойство уведомления `object` хранит окончательный вариант текста.

Поскольку текстовое поле для ввода текста использует клавиатуру, то уведомления о действиях могут быть отправлены, когда произойдет еще и одно из следующих событий:

- `UIKeyboardWillShowNotification` отправляет до момента отображения клавиатуры;
- `UIKeyboardDidShowNotification` отправляет после отображения клавиатуры;
- `UIKeyboardWillHideNotification` отправляет до момента скрытия клавиатуры;
- `UIKeyboardDidHideNotification` отправляет после скрытия клавиатуры.

Прокрутка текстовых полей

При редактировании текстового поля, расположенного около нижней границы экрана, всплывающая клавиатура может иногда скрывать само поле. Начиная с версии 2.2 операционной системы iPhone была автоматизирована прокрутка. Для решения этой проблемы в более ранних версиях встроенного программного обеспечения можно прокрутить вверх представление, содержащее элемент управления, чтобы перенести текстовое поле в область видимости. Эта функциональность использует два метода-делегата текстового поля: `textFieldDidBeginEditing` и `textFieldShouldReturn`.

Для данного текстового поля назначьте в качестве делегата класс представления, содержащий это поле:

```
textControl.delegate = self;
```

Когда пользователь касается текстового поля, уведомляется метод-делегат `textFieldDidBeginEditing`. Используйте этот факт для изменения начала координат и размера класса представления на фиксированное число. Вы можете анимировать это действие, чтобы создать эффект прокрутки; Core Animation управляет всеми фреймами в процессе построения промежуточных отображений ("tweiining"), поэтому вы может просто сказать этой библиотеке, где вы хотите, чтобы представление остановилось, а Core Animation выполнит всю промежуточную работу.

Приведенный далее код прокручивает представление вверх на 240 пикселов, когда пользователь касается текстового поля:

```
- (void)textFieldDidBeginEditing:(UITextView *)textview {
    [ UIView beginAnimations: nil context: NULL ];
    [ UIView setAnimationDuration: 0.3 ];

    CGRect frame = self.view.frame;
    frame.origin.y -= 240.0;
    frame.size.height += 240.0;
    self.view.frame = frame;

    [ UIView commitAnimations ];
}
```

Когда пользователь нажимает кнопку возврата, фрейм может быть прокручен в прежнее положение. Также вы можете отменить клавиатуру, оставив первый отвечающий объект. Такой же тип транзакции создается для анимации этого изменения и создания эффекта прокрутки:

```
- (BOOL)textFieldShouldReturn:(UITextView *) textView {
    [ UIView beginAnimations: nil context: NULL ];
    [ UIView setAnimationDuration: 0.3 ];

    CGRect frame = self.view.frame;
    frame.origin.y += 240.0;
    frame.size.height -= 240.0;
    self.view.frame = frame;
```

```
    [ UIView commitAnimations ];
    [ textView resignFirstResponder ];
    return YES;
}
```

Если вы в одном и том же представлении используете несколько текстовых полей, то может потребоваться сравнить указатель `UITextView`, передаваемый в этот метод, чтобы вы могли выполнить различные прокрутки в зависимости от расположения текстового поля и исходя из положений полос прокрутки. Приведенный только что пример использует значение прокрутки `240.0`, однако вам потребуются различные значения в зависимости от того, где на экране расположены ваши текстовые поля. Более подробную информацию о прокрутке вы найдете в *разд. "Представления прокрутки" далее в этой главе*.

Кнопки

Кнопки (buttons) являются простыми элементами управления, отображающими либо текст, либо изображение, которые уведомляют приложение при нажатии на них. Вы можете прикрепить элементы управления типа "кнопка" к объектам `UIView`, ячейкам таблиц и целому ряду других объектов. Поскольку они порождены от класса `UIControl`, то используют одну и ту же структуру уведомлений, что и базовый класс. Кроме этого, класс `UIButton` предоставляет целый ряд дополнительных функциональных возможностей.

Создание элемента управления

Элемент управления типа кнопка является стандартным объектом `UIControl`. Вы можете инициализировать его, вызвав метод элемента управления `initWithFrame`, чтобы задать начало координат и ширину данного элемента управления. Кроме того, класс `UIButton` предоставляет статический метод `buttonWithType` для быстрого создания целого ряда предопределенных стилей:

```
UIButton *myButton = [UIButton buttonWithType: UIButtonTypeRoundedRect];
```

В качестве аргумента типа с методом `buttonWithType` вы можете предоставить следующие стили кнопок (табл. 10.2).

Таблица 10.2

Стиль	**Описание**
`UIButtonTypeCustom`	Пользовательская кнопка; без стиля
`UIButtonTypeRoundedRect`	Белая кнопка с закругленными углами, наподобие ячейки таблицы настроек или карточки адресной книги
`UIButtonTypeDetailDisclosure`	Синее раскрытие, располагающееся рядом с любым текстом
`UIButtonTypeInfoLight`	Информационный круг, располагающийся рядом с любым текстом, для виджетов
`UIButtonTypeInfoDark`	Темный информационный круг для белого фона
`UIButtonTypeContactAdd`	Синяя кнопка со знаком "плюс" (+), располагающаяся рядом с любым текстом

После создания кнопки вы можете задать ее начало координат и размер, назначив структуру `CGRect` свойству фрейма кнопки:

```
CGRect *myButtonFrame = CGRectMake(25.0, 25.0, 100.0, 100.0);
myButton.frame = myButtonFrame;
```

Можно определить названия кнопок для любого заданного состояния кнопки. Для этого воспользуйтесь методом `setTitle`:

```
[ myButton setTitle: @"Click Here" forState: UIControlStateNormal ];
```

Вы также можете определить изображения на кнопках для заданного состояния кнопки. Для этого воспользуйтесь методом `setImage`:

```
[ myButton setImage:
    [ UIImage applicationImageNamed: @"button.png" ]
    forState: UIControlStateNormal
];
```

Кроме того, для каждого состояния кнопки вы можете задать цвет и тень для названия, а также фон кнопки. Метод `setTitleColor` и метод `setTitleShadowColor` требуют в качестве аргумента объект `UIColor`. Метод `setBackgroundImage` в качестве аргумента требует объект `UIImage`:

```
[ myButton setTitleColor:
    [ UIColor redColor ] forState: UIControlStateNormal ];
```

```
[ myButton setTitleShadowColor:
    [ UIColor grayColor ] forState: UIControlStateNormal ];

[ myButton setBackgroundImage:
    [ UIImage applicationImageNamed: @"background.png" ]
    forState: UIControlStateNormal ];
```

Каждый из пяти методов, о которых вы только что узнали, содержит аргумент `forState`. Он определяет, для какого состояния кнопки отображаются те или иные заголовки, изображения и другие свойства. Вы можете запрограммировать кнопку на изменение ее внешнего вида при изменении состояния. Эти состояния — те же самые, с которыми вы уже познакомились в классе `UIControl`:

- `UIControlStateNormal`;
- `UIControlStateHighlighted`;
- `UIControlStateDisabled`;
- `UIControlStateSelected`.

Если кнопка подсвечена или запрещена, то класс `UIButton` может настроить ее внешний вид. Приведенные далее свойства позволят вам определить, как должна выглядеть кнопка:

- `adjustsImageWhenHighlighted` используется по умолчанию, если кнопка нажата, то изображение подсвечивается. Чтобы запретить эту функциональность, установите это свойство в `NO`;
- `adjustImageWhenDisabled` используется по умолчанию, если кнопка запрещена, то изображение отображается более темным. Чтобы запретить эту функциональность, установите это свойство в `NO`;
- `showsTouchWhenHighlighted`. Чтобы кнопка при нажатии светилась, установите это свойство в `YES`. Это полезно для информационных кнопок или кнопок, являющихся по тем или иным причинам важными.

Отображение элемента управления

После инициализации кнопки отобразите ее, добавив как подчиненное представление к объекту любого типа, который может его разместить. Далее приведен соответствующий пример:

```
[ myView addSubview: button ];
```

Визуализация подмен

Помимо многочисленных стилевых параметров объекта `UIButton` для настройки визуализации кнопки вы можете переопределить ряд методов в подклассах объекта `UIButton`. Эти методы возвращают структуру `CGRect`, определяющую границы для каждого компонента кнопки. Если вы создаете пользовательский класс на базе `UIButton`, то можете переопределить эти методы, чтобы изменить одну или несколько границ. Никогда не вызывайте эти методы напрямую. Далее приведен соответствующий пример:

```
- (CGRect) imageRectForContentRect: (CGRect) bounds {
    return myButtonImageBounds;
}
```

При создании подкласса `UIButton` доступны следующие подмены:

- `backgroundRectForBounds` — границы, в пределах которых визуализировать фоновое изображение;
- `contentRectForBounds` — границы, в пределах которых визуализировать содержимое кнопки;
- `titleRectForContentRect` — границы, в пределах которых визуализировать текст названия кнопки;
- `imageRectForContentRect` — границы, в пределах которых визуализировать изображение кнопки.

Страницы

Страницы (page controls) предоставляют визуальный индикатор для приложений, которые перелистывают виртуальные страницы с помощью большого пальца, а не кнопок навигации, или нуждаются в индикаторе страниц по каким-либо другим причинам. Вы узнаете о перелистывании страниц в *главе 13*, описывающей данный элемент управления. Элемент управления "страница" отображается в виде ряда точек вдоль верхней или нижней границы экрана и обновляется вашим приложением по мере перелистывания пользователем страниц. Чаще всего их можно встретить вдоль нижней границы домашнего экрана iPhone (springboard) при добавлении слишком большого количества значков, которые не помещаются на одном экране, а также в Safari, когда открыто окно выбора страниц. Элементы управления страницами подходят для собственных классов представлений, в которых разработчик старается отобразить информацию сквозь несколько страниц.

Создание элемента управления

Чтобы создать элемент управления страницами, воспользуйтесь стандартным методом `initWithFrame`, чтобы задать начало координат и размер данного элемента управления. Если задать нулевой размер, то размер по горизонтали и размер по вертикали будут автоматически заданы исходя из количества страниц. Задание в качестве размера ширины экрана приведет к центровке элемента управления по горизонтали в пределах области отображения:

```
UIPageControl *pageControl = [ [ UIPageControl alloc ]
    initWithFrame: CGRectMake(0.0, 400.0, 320.0, 0.0) ];
```

Чтобы задать количество страниц, которые будет заключать элемент управления, задайте свойство `numberOfPages`:

```
pageControl.numberOfPages = 5;
```

По умолчанию выбрана первая страница. Чтобы выбрать другую страницу, задайте свойство `currentPage`. Нумерация страниц начинается с нуля, поэтому первая страница задана как 0:

```
pageControl.currentPage = 0;
```

По умолчанию индикатор отображается даже, если имеется только одна страница. Чтобы скрыть индикатор при наличии только одной страницы, установите свойство `hidesForSinglePage` в `NO`:

```
pageControl.hidesForSinglePage = NO;
```

Наконец, если вы хотите, чтобы индикатор не обновлял текущую страницу, а вы имели время на выполнение ваших собственных действий, то установите свойство `defersCurrentPageDisplay` в `YES`. Для этого вам потребуется вызывать элемент управления, чтобы обновить текущую страницу:

```
pageControl.defersCurrentPageDisplay = YES;
[ pageControl updateCurrentPageDisplay ];
```

Отображение элемента управления

После инициализации элемента управления "страница" отобразите его, добавив как подчиненное представление к объекту любого типа, который может его разместить. Далее приведен соответствующий пример:

```
[ myView addSubview: pageControl ];
```

Страница настроена на использование прозрачного фона, поэтому если вы хотите, чтобы элемент управления отвечал на касания, как он это делает в Springboard, то вам потребуется убедиться в том, что за ним имеется еще один объект представления.

Уведомления

Когда пользователь касается страницы, для `UIControlEventValueChanged` создается событие. Вы можете задать действие с помощью метода `addTarget` класса `UIControl`:

```
[ myView addTarget: self action:@selector(pageChanged:)
    forControlEvents: UIControlEventValueChanged
];
```

Затем уведомляется метод действия, который может считать значение новой страницы и предпринять соответствующее действие:

```
- (void) pageChanged: (id) selector {
    UIPageControl *control = (UIPageControl *) selector;
    NSInteger page = control.currentPage;

    /* Код для обработки изменения страницы */
}
```

Для дальнейшего изучения

Теперь, когда вы познакомились с элементами управления, попробуйте использовать их в своем коде.

- Создайте приложение с использованием различных элементов управления. Прикрепите элементы управления к классам `UIView`, панелям навигации и другим объектам. К каким объектам вы можете прикрепить элементы управления?
- В заголовочных файлах вашего SDK проверьте наличие прототипов UIControl.h, UISwitch.h, UISlider.h, UIButton.h, UITextField.h и UISegmentedControl.h. Вы найдете их в папке /Developer/Platforms/iPhoneOS.platform внутри каталога Headers библиотеки UI Kit.

Таблицы настроек

Таблицы настроек (preferences tables) предоставляют эстетически богатый интерфейс для отображения изменения программных настроек или отображения структурированной информации, например, сетевой информации или информации о контактах. Когда это возможно, приложение должно использовать *блок настроек* (preference bundle), обсуждаемый в *главе 11*, чтобы добавить вкладку с настройками в приложение iPhone Settings, но недостатком этого является то, что для изменения настроек требуется выйти из вашего приложения. Apple предоставила санкционированный API для создания таблиц настроек и в ваших приложениях, что полезно для изменения параметров среды исполнения. Помимо настроек, если ваше приложение отображает таблицу сгруппированной информации, например, собственное отображение контактов или сетевую информацию, то этот тип таблиц также может оказаться полезным для упорядочивания ваших данных. Таблицы настроек предоставляют ячейки изменяемого размера, способные вмещать элементы управления, текстовые поля и информационный текст. Они также предоставляют механизм для логического объединения схожих настроек.

SDK удобно обернул класс таблиц настроек в существующий класс `UITableView`, о котором вы узнали в *главе 3*. В результате вы можете создавать таблицу настроек точно так же, как и стандартную таблицу с минимальными корректировками ее стиля и логических группировок.

> ***Примечание***
>
> Если погрузиться глубже, то можно увидеть, что взаимодействие с классом `UITableView` порождает низкоуровневый объект `UIPreferencesTable`. Этот объект скрыт от SDK, однако широко используется разработчиками открытого кода, пишущими код с применением пакета инструментов сторонних фирм. SDK сделал гораздо более удобной работу со всеми основными структурами таблиц, консолидировав интерфейсы для них в класс `UITableView` — единственный класс, который вам надо изучить.

Создание таблицы настроек

При реализации таблицы настроек нужно быть достаточно предусмотрительным, поскольку для запроса информации, необходимой для заполнения таблицы, используется *источник данных* (data source). Это делается аналогично общей реализации класса `UITableView`, изученного вами в *главе 3*, но на бо-

лее высоком уровне сложности. Исполняющий класс вызывает в источнике данных целый ряд методов протокола для возврата информации о таблице настроек, точно так же, как и для стандартной таблицы. К большому разочарованию сообщества разработчиков приложений для iPhone все это существенно отличается от объектно-ориентированной модели. Вместо этого создание всей таблицы настроек является весьма трудоемким и сложным занятием в противовес традиционно элегантному стилю проектирования Apple.

В качестве резюме стоит сказать, что таблица настроек ссылается на страницу настроек или на информационную страницу. 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 необязательный заголовок, текст и элементы управления, если они требуются.

Рис. 10.1. Аналогия диалога между таблицей настроек и ее источником данных

Диалог между таблицей настроек и ее источником данных выглядит примерно так, как показано на рис. 10.1.

Поскольку таблица настроек состоит из двух частей (сама таблица и источник данных), то самый правильный способ собрать все воедино — это создать подкласс `UITableViewController`, как вы делали это в *главе 3*, и заставить его выступать в роли собственного источника данных. Это позволит вашему приложению просто создавать экземпляр класса контроллера и отображать его на экране.

Создание подклассов контроллера представления таблицы

Чтобы создать автономную таблицу настроек, создайте подкласс `UITableViewController` и включите все методы протокола, необходимые для привязки к ней самой как к источнику данных. В приведенном далее примере создается подкласс `MyPreferencesViewController`:

```
@interface MyPreferencesViewController : UITableViewController
{

}

/* Методы таблицы настроек */

- (id)init;
- (void)loadView;
- (void)dealloc;

/* Методы источника данных */

- (NSInteger)numberOfSectionsInTableView:(UITableView *)tableView;
- (NSInteger)tableView:(UITableView *)tableView
    numberOfRowsInSection:(NSInteger)section;
- (UITableViewCell *)tableView:(UITableView *)tableView
    cellForRowAtIndexPath:(NSIndexPath *)indexPath;
- (CGFloat)tableView:(UITableView *)tableView
    heightForRowAtIndexPath:(NSIndexPath *)indexPath;
- (NSString *)tableView:(UITableView *)tableView
    titleForHeaderInSection:(NSInteger)section;
```

Методами, используемыми для работы с источником данных, являются следующие:

- `numberOfSectionsInTableView` возвращает количество логических групп в таблице настроек. Метки групп не должны включаться в подсчет групп. Каждая логическая группа отображается на экране как отдельный "пузырь" (balloon);
- `numberOfRowsInSection` возвращает количество строк в заданной группе настроек. Подсчет строк не должен включать метки групп. Ваш источник данных будет рассматривать каждую строку как объект `UITableViewCell`, возвращаемый методом `cellForRowAtIndexPath`;
- `cellForRowAtIndexPath` возвращает объект `UITableViewCell`, соответствующий указанным группе и строке. Этот метод должен проверять наличие в очереди существующей ячейки точно так же, как в примере TableDemo из *главы 3*;
- `titleForHeaderInSection` возвращает объект `NSString`, содержащий метку группы для заданной группы настроек;
- `heightForRowAtIndexPath` возвращает пользовательскую высоту для указанных группы и строки. Это позволяет настраивать высоту определенных ячеек.

Инициализация таблицы

Когда вы инициализируете контроллер табличного представления, внутри класса создается внутренний объект `UITableView`, чтобы представлять табличное представление, принадлежащее контроллеру. Поскольку вы создаете особый тип таблицы (таблицу настроек), то вам придется создать контроллер табличного представления с другим стилем таблицы. Подмените метод `init` вашего класса `MyPreferencesViewController` и вручную создайте табличное представление с помощью альтернативного метода `initWithStyle` суперкласса `UITableViewController`:

```
(id)init {
    self = [ super initWithStyle: UITableViewStyleGrouped ];

    if (self != nil) {
        /* Код инициализации */
    }
    return self;
}
```

Если вы работаете непосредственно с `UITableView` (т. е. без контроллера табличного представления), то можете использовать альтернативный метод `initWithFrame` класса табличного представления, который включает аргумент стиля. Вот соответствующий пример:

```
UITableView *myTableView = [ [ UITableView alloc ]
    initWithFrame: myRect
    style: UITableViewStyleGrouped
];
```

После создания объекта вызовите его метод `reloadData` для загрузки всех элементов в таблицу настроек. В результате данная таблица вызовет свой источник данных и начнет загрузку информации о группировке и конфигурации ячеек. Вы можете в любой момент вручную перезагрузить таблицу, вызвав метод `reloadData` класса таблицы:

```
[ self.tableView reloadData ];
```

Ячейки таблицы настроек

Каждая ячейка в таблице настроек создается как объект `UITableViewCell` или же его подкласс. Ячейки возвращаются посредством метода обратного вызова `cellForRowAtIndexPath`, который вы должны написать и который автоматически вызывается классом таблицы настроек, как только новые строки отрисовываются на экране. Например, если ваша таблица вызывает метод `cellForRowAtIndexPath`, задавая `[0 1]`, то данный метод может возвращать ячейку, соответствующую первой группе (группа 0) и второй строке (строка 1). Данный метод должен проверять очередь, чтобы удостовериться в том, что такая ячейка еще не была создана, а затем создавать и возвращать ее:

```
- (UITableViewCell *)tableView:(UITableView *)tableView
    cellForRowAtIndexPath:(NSIndexPath *)indexPath
{

    NSString *CellIdentifier = [ fileList objectAtIndex:
        [ indexPath indexAtPosition: 1 ]
    ];

    UITableViewCell *cell = [ tableView
        dequeueReusableCellWithIdentifier: CellIdentifier
    ];
```

```
    /* Создаем ячейку с нуля */
    if (cell == nil) {
        cell = [ [ [ UITableViewCell alloc ]
           initWithFrame: CGRectZero
           reuseIdentifier: CellIdentifier
         ] autorelease ];

        /* Пример содержимого ячейки */
        cell.text = @"Debugging";
    }

    /* Возвращаем либо новую ячейку, либо ячейку из очереди */
    return cell;
}
```

Элементы управления

Класс `UITableViewCell` может вмещать целый ряд элементов управления, позволяя вам добавлять к отдельным ячейкам таблицы переключатели, бегунки или элементы управления других типов. Элементы управления добавляются к объекту `UITableViewCell` как подчиненные представления.

При создании новой ячейки таблицы настроек добавление элемента управления осуществляется с помощью метода `addSubview` ячейки:

```
cell = [ [ [ UITableViewCell alloc ] initWithFrame: CGRectZero
            reuseIdentifier: CellIdentifier
          ] autorelease ];

cell.text = @"Advanced Mode";

/* Добавляем переключатель к ячейке */
UISwitch *debugSwitch = [ [ UISwitch alloc ]
    initWithFrame: CGRectMake(200.0, 10.0, 0.0, 0.0)
];

/* Прикрепляем переключатель к ячейке */
[ cell addSubview: debugSwitch ];
```

В приведенном примере создается элемент управления типа "переключатель" со сдвигом фрейма вправо от ячейки. Данный пример также создает ячейку с текстом "Advanced Mode" и прикрепляет к ней переключатель. Все это происходит в методе `cellForRowAtIndexPath` до возвращения только что созданной ячейки.

Текстовые поля

Вы можете добавлять текстовые поля (text fields) точно так же, как и другие элементы управления. С помощью системы уведомлений класса `UIControl` событие редактирования и другие релевантные события могут передаваться в ваш класс-делегат. Текстовые поля включают в себя смещение и область отображения, задающие размер текстового поля. Это позволяет вам создавать либо специальные текстовые ячейки (отображающие только текст внутри всей ячейки), либо ячейку с заголовком, выделенным полужирным шрифтом (задается свойством `property`), и с текстовым полем:

```
UITextField *textField = [ [ UITextField alloc ]
    initWithFrame: CGRectMake(20.0, 10.0, 280.0, 50.0)
];


[ cell addSubview: textField ];
```

Также вы можете отображать текст и делать его нередактируемым с помощью метода `setEnabled`:

```
textField.text = @"Some text";
[ cell setEnabled: NO ];
```

С помощью того же самого свойства вы можете считать значение:

```
NSString *text = textField.text;
```

Отображение таблицы настроек

Таблица настроек может быть отображена, как и контроллер представления, путем прикрепления ее представления к окну или посредством проталкивания ее в контроллер представлений. Чтобы задать активным представление для окна, используйте следующий код:

```
[ window addSubview: myTableViewController.view ];
```

Чтобы протолкнуть таблицы в контроллер навигации, используйте следующий код:

```
[ navigationController pushViewController: myTableViewController.view
        animated: YES
];
```

Пример таблицы настроек: ShootStuffUp

Вы пишете космическую игру-стрелялку, и ей необходим набор параметров для управления всем: от уровня громкости звука до отладочных сообщений. В данном примере класс `UITableViewController` переопределяется для создания пользовательского объекта `ShootStuffUpTableViewController` (рис. 10.2). Этот объект содержит свой источник данных для собственной табличной структуры. Он создает каждую ячейку и назначает некоторые из элементов управления, о которых вы узнали в предыдущем разделе. Объект `MyPreferencesTable` построен как независимая таблица настроек, которая может быть использована классом `MainView`.

Рис. 10.2. Пример ShootStuffUp

Вы можете скомпилировать это приложение, приведенное в листингах 10.1—10.5, с помощью SDK, создав проект приложения ShootStuffUp на базе представления. Если вы хотите увидеть, как создавать все эти объекты с нуля, то удалите код Interface Builder.

Листинг 10.1. Прототипы делегата приложения ShootStuffUp (ShootStuffUpAppDelegate.h)

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

@class ShootStuffUpViewController;

@interface ShootStuffUpAppDelegate : NSObject <UIApplicationDelegate> {
    UIWindow *window;
    ShootStuffUpViewController *viewController;
    UINavigationController *navigationController;
}

@property (nonatomic, retain) IBOutlet UIWindow *window;
@property (nonatomic, retain) IBOutlet ShootStuffUpViewController
  *viewController;

@end
```

Листинг 10.2. Делегат приложения ShootStuffUp (ShootStuffUpAppDelegate.m)

```
#import "ShootStuffUpAppDelegate.h"
#import "ShootStuffUpViewController.h"

@implementation ShootStuffUpAppDelegate

@synthesize window;
@synthesize viewController;

- (void)applicationDidFinishLaunching:(UIApplication *)application {
    CGRect screenBounds = [ [ UIScreen mainScreen ] bounds ];
```

```
    self.window = [ [ [ UIWindow alloc ] initWithFrame: screenBounds ]
      autorelease ];
    viewController = [ [ ShootStuffUpViewController alloc ] init ];
    navigationController = [ [ UINavigationController alloc ]
initWithRootViewController: viewController ];

    [ window addSubview: [ navigationController view ] ];
    [ window makeKeyAndVisible ];
}

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

@end,
```

Листинг 10.3. Прототипы контроллера табличного представления (ShootStuffUpViewController.h)

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

@interface ShootStuffUpViewController : UITableViewController {
    UISlider *musicVolumeControl;
    UISlider *gameVolumeControl;
    UISegmentedControl *difficultyControl;

    UISlider *shipStabilityControl;
    UISwitch *badGuyControl;
    UISwitch *debugControl;

    UITextField *versionControl;
}

- (id) init;
```

```
- (void) dealloc;
- (NSInteger)numberOfSectionsInTableView:(UITableView *)tableView;
- (NSInteger)tableView:(UITableView *)tableView numberOfRowsIn
Section:(NSInteger)section;
- (NSString *)tableView:(UITableView *)tableView titleForHeader
InSection:(NSInteger)section;
- (UITableViewCell *)tableView:(UITableView *)tableView cellForRow
AtIndexPath:(NSIndexPath *)indexPath;

@end
```

Листинг 10.4. Контроллер табличного представления ShootStuffUp (ShootStuffUpViewController.m)

```
#import "ShootStuffUpViewController.h"

@implementation ShootStuffUpViewController

- (id) init {
    self = [ super initWithStyle: UITableViewStyleGrouped ];

    if (self != nil) {
        self.title = @"Game Settings";
    }
    return self;
}

- (void) loadView {
    [ super loadView ];
}

- (BOOL)shouldAutorotateToInterfaceOrientation:
    (UIInterfaceOrientation)interfaceOrientation
{
    return (interfaceOrientation == UIInterfaceOrientationPortrait);
}
```

```
- (void)didReceiveMemoryWarning {

    [ super didReceiveMemoryWarning ];
}

- (void)dealloc {
    [ musicVolumeControl release ];
    [ gameVolumeControl release ];
    [ difficultyControl release ];
    [ shipStabilityControl release ];
    [ badGuyControl release ];
    [ debugControl release ];
    [ versionControl release ];
    [ super dealloc ];
}

- (NSInteger)numberOfSectionsInTableView:(UITableView *)tableView {
    return 3;
}

- (NSInteger)tableView:(UITableView *)tableView
    numberOfRowsInSection:(NSInteger)section
{
    switch (section) {
        case(0):
            return 3;
            break;
        case(1):
            return 3;
            break;
        case(2):
            return 1;
            break;
    }

    return 0;
}
```

```
- (NSString *)tableView:(UITableView *)tableView
    titleForHeaderInSection:(NSInteger)section
{
    switch (section) {
        case(0):
            return @"Game Settings";
            break;
        case(1):
            return @"Advanced Settings";
            break;
        case(2):
            return @"About";
            break;
    }
    return nil;
}

- (UITableViewCell *)tableView:(UITableView *)tableView
    cellForRowAtIndexPath:(NSIndexPath *)indexPath
{
    NSString *CellIdentifier = [ NSString stringWithFormat: @"%d:%d",
        [ indexPath indexAtPosition: 0 ], [ indexPath indexAtPosition:1 ]
    ];

    UITableViewCell *cell = [ tableView
        dequeueReusableCellWithIdentifier: CellIdentifier
    ];

    if (cell == nil) {
      cell = [ [ [ UITableViewCell alloc ]
        initWithFrame: CGRectZero reuseIdentifier: CellIdentifier
      ] autorelease ];

      cell.selectionStyle = UITableViewCellSelectionStyleNone;

      switch ([ indexPath indexAtPosition: 0]) {
        case(0):
```

```
switch([ indexPath indexAtPosition: 1]) {
  case(0):
    musicVolumeControl = [ [ UISlider alloc ]
      initWithFrame: CGRectMake(170, 0, 125, 50) ];
    musicVolumeControl.minimumValue = 0.0;
    musicVolumeControl.maximumValue = 10.0;
    musicVolumeControl.value = 3.5;
    [ cell addSubview: musicVolumeControl ];
    cell.text = @"Music Volume";
    break;
  case(1):
    gameVolumeControl = [ [ UISlider alloc ]
      initWithFrame: CGRectMake(170, 0, 125, 50) ];
    gameVolumeControl.minimumValue = 0.0;
    gameVolumeControl.maximumValue = 10.0;
    gameVolumeControl.value = 3.5;
    [ cell addSubview: gameVolumeControl ];
    cell.text = @"Game Volume";
    break;
  case(2):
    difficultyControl = [ [ UISegmentedControl alloc ]
      initWithFrame: CGRectMake(170, 5, 125, 35) ];
    [ difficultyControl insertSegmentWithTitle: @"Easy"
      atIndex: 0 animated: NO ];
    [ difficultyControl insertSegmentWithTitle: @"Hard"
      atIndex: 1 animated: NO ];
    difficultyControl.selectedSegmentIndex = 0;
    [ cell addSubview: difficultyControl ];
      cell.text = @"Difficulty";
    break;
  }
  break;
case(1):
  switch ([ indexPath indexAtPosition: 1 ]) {
    case(0):
      shipStabilityControl = [ [ UISlider alloc ]
        initWithFrame: CGRectMake(170, 0, 125, 50) ];
```

```
                shipStabilityControl.minimumValue = 0.0;
                shipStabilityControl.maximumValue = 10.0;
                shipStabilityControl.value = 3.5;
                [ cell addSubview: shipStabilityControl ];
                cell.text = @"Ship Stability";
                break;
              case(1):
                badGuyControl = [ [ UISwitch alloc ]
                  initWithFrame: CGRectMake(200, 10, 0, 0) ];
                badGuyControl.on = YES;
                [ cell addSubview: badGuyControl ];
                cell.text = @"Bad Guys";
                break;
              case(2):
                debugControl = [ [ UISwitch alloc ]
                  initWithFrame: CGRectMake(200, 10, 0, 0) ];
                debugControl.on = NO;
                [ cell addSubview: debugControl ];
                cell.text = @"Debug";
                break;
              }
              break;
            case(2):
              versionControl = [ [ UITextField alloc ]
                initWithFrame: CGRectMake(170, 10, 125, 38) ];
              versionControl.text = @"1.0.0 Rev. B";
              [ cell addSubview: versionControl ];
              [ versionControl setEnabled: NO ];
              cell.text = @"Version";
              break;
          }
      }
      return cell;
}

@end
```

Листинг 10.5. Функция `main` для ShootStuffUp (main.m)

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

int main(int argc, char *argv[]) {
  NSAutoreleasePool * pool = [ [ NSAutoreleasePool alloc ] init ];
  int retVal = UIApplicationMain(argc, argv, nil,
                                 @"ShootStuffUpAppDelegate");
  [ pool release ];
  return retVal;
}
```

Как это работает

Вы только что познакомились с полноценным приложением, отображающим таблицу настроек. Вот как оно работает:

1. Когда вы запускаете приложение, класс-делегат создает подкласс класса `UITableViewController`. Этот класс включает в себя источник данных для таблицы настроек и переменное хранилище для элементов управления.
2. Метод `init` контроллера табличного представления подменяется и указывает своему суперклассу создать его с использованием стиля таблицы настроек — `UITableViewStyleGrouped`.
3. Контроллер табличного представления добавляется в стек контроллера навигации, а затем отображается, вызывается его метод `reloadData`. Поскольку мы не подменяли `reloadData`, то вызывается версия метода родительского `UITableView` класса. Это инициирует начало взаимодействия с источником данных посредством вызова различных методов источника данных. Таблица настроек общается с источником данных, чтобы определить основную конструкцию и геометрию таблицы и ее элементов управления.
4. Метод `cellForRowAtIndex` сначала проверяет, не существует ли уже такая ячейка в очереди памяти таблица. Если не существует, то создается новый объект `UITableViewCell`. Задаются заголовок ячейки и ее элементы управления исходя из номера строки и группы, а затем создаются все элементы управления и добавляются как подчиненные представления ячейки настроек.

Для дальнейшего изучения

Теперь, когда вы получили представление о том, как работают таблицы настроек, попробуйте выполнить некоторые упражнения для более близкого знакомства с ними.

- Воспользуйтесь своими знаниями системы уведомлений класса `UIControl` для перехвата изменений значения, происходящих в режиме реального времени.
- Воспользуйтесь примерами из *главы 3* для создания основного представления для данного приложения, которое отображает все значения предпочтения. Используйте навигационный элемент управления, чтобы пользователь мог перемещаться между таблицей настроек и вашим отображаемым представлением.

Списки разделов

Класс `UITableView` является достаточно многогранным. Он включает в себя не только стандартные таблицы списочного типа и таблицы настроек, но еще и таблицы другого типа, широко используемого в программном обеспечении iPhone: списки разделов (section lists). Когда таблица становится достаточно большой, то нахождение в ней какого-либо определенного элемента становится сродни поиску иголки в стоге сена. Список разделов предоставляет визуальную структуру, аналогичную стандартной таблице, но расширенную и включающую в себя отдельные группировки ячеек и индекс в стиле старой картотеки для быстрого перехода к заголовку раздела. Каждой группировке может быть сопоставлен заголовок раздела наподобие жанра книг или первой букве контакта. Списки разделов применяются в собственных контактах iPhone и списках песен.

Как и другие таблицы, список разделов использует источник данных. Источник данных (data source) — это протокольный интерфейс для выполнения запросов к объекту на предмет содержимого и структуры таблицы. Источник данных для списка разделов предоставляет протокольные методы, необходимые для построения группировок списка разделов, заголовков разделов, индекса и отдельных ячеек строк. Как и в случае с другими табличными классами, описываемыми в этой книге, приводимые здесь примеры создают подкласс объекта `UITableViewController`, который может содержать как таблицу разделов, так и соответствующий источник данных. Такая архитек-

тура является наиболее наглядной и лучше всего подходит для создания специализированных повторно используемых классов.

> ***Примечание***
>
> Если опуститься на более низкий уровень данной библиотеки, то там происходит следующее. UI Kit порождает класс `UISectionList`, который инкапсулирует `UISectionTable`, содержащий табличную часть списка. Как и в случае с таблицами настроек, эти низкоуровневые объекты скрыты от разработчика, поэтому для построения данного объекта вы снова будете работать со стандартными интерфейсами класса `UITableView`.

Создание списка разделов

Списки разделов создаются практически точно так же, как и таблица настроек, и для этого используются те же самые подмены. Тем не менее, в конструкциях списка разделов и таблицы настроек имеются два отличия: структура основной таблицы и дополнительные методы-делегаты для добавления индексной панели.

На самом деле эти два табличных формата настолько похожи, что простая замена приведенной далее строчки кода в примере ShootStuffUp из предыдущего раздела отобразит вашу таблицу настроек как список разделов. Замените этот код:

```
self = [ super initWithStyle: UITableViewStyleGrouped ];
```

на этот:

```
self = [ super init ];
```

Когда вы создаете основную таблицу, ее стиль по умолчанию разрешает разделы и индексацию (рис. 10.3). Заменив приведенную выше строчку кода на вызов `init`, вы вернули стиль таблицы в его заданный по умолчанию стиль, создаваемый автоматически, если это не сделано вручную.

Ваш контроллер табличного представления будет использовать те же самые методы-делегаты, что и таблица представления. Вот соответствующий пример:

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


@interface MySectionListViewController : UITableViewController {
}
```

```
- (id) init;
- (void) dealloc;
- (NSInteger)numberOfSectionsInTableView:(UITableView *)tableView;
- (NSInteger)tableView:(UITableView *)tableView
    numberOfRowsInSection:(NSInteger)section;
- (NSString *)tableView:(UITableView *)tableView
    titleForHeaderInSection:(NSInteger)section;
- (UITableViewCell *)tableView:(UITableView *)tableView
    cellForRowAtIndexPath:(NSIndexPath *)indexPath;

@end
```

Рис. 10.3. ShootStuffUp в виде списка разделов

Если вы будете добавлять индексную панель (панель в стиле старой картотеки, которая имеется в вашем списке контактов iPhone), то для получения элементов индекса вам придется добавить еще одну дополнительную подмену:

```
- (NSArray *)sectionIndexTitlesForTableView:(UITableView *)tableView;
```

Методами, используемыми для источника данных, являются следующие:

- `numberOfSectionsInTableView` возвращает количество разделов для отображения в таблице настроек. Пустые разделы включаются сюда, только если вы хотите, чтобы они отображались пустыми;
- `numberOfRowsInSection` возвращает количество строк в заданном разделе;
- `cellForRowAtIndexPath` возвращает объект `UITableViewCell`, соответствующий заданным разделу и строке. Этот метод должен проверять наличие ячейки в очереди точно так же, как это делалось в примере TableDemo из *главы 3*;
- `titleForHeaderInSection` возвращает объект `NSString`, содержащий метку заданного раздела. Это может быть как один символ (как, например, в списке контактов), так и целая строка (как, например, жанры книг);
- `heightForRowAtIndexPath` возвращает пользовательскую высоту заданных группы и строки. Это позволяет настраивать высоту определенных ячеек;
- `sectionIndexTitlesForTableView` возвращает объект `NSArray`, содержащий массив объектов `NSString`, необходимых для построения индексной панели;
- `sectionForSectionIndexTitle` ставит в соответствие заданный элемент индекса номеру раздела, чтобы касание индекса переносило на экран соответствующий раздел. Это особенно удобно при создании неалфавитных индексов, например, жанров книг и т. п.

Добавление индексной панели

Когда к списку разделов вы добавляете индексную панель (index bar), то справа появляется панель в стиле старой картотеки (Rolodex-style), позволяющая пользователю быстро выбирать нужный раздел, щелкая по соответствующему индексу. Обычно заголовки индекса расположены в алфавитном порядке, но индексную панель можно настроить на использование любого набора заголовков.

Чтобы добавить индексную панель, добавьте метод источника данных `sectionindexTitlesForTableView`. Вот соответствующий пример:

```
- (NSArray *)sectionIndexTitlesForTableView:(UITableView *)tableView {
    return [ NSMutableArray arrayWithObjects:
         @"A", @"B", @"C", @"D", @"E", @"F",
```

```
        @"G", @"H", @"I", @"J", @"K", @"L",
        @"M", @"N", @"O", @"P", @"Q", @"R",
        @"S", @"T", @"U", @"V", @"W", @"X",
        @"Y", @"Z", @"#", nil
    ];
}
```

Отображение списка разделов

Список разделов отображается точно так же, как и стандартный контроллер табличного представления, — посредством прикрепления его основного представления к окну или путем проталкивания его в контроллер навигации.

Чтобы задать активное представление для окна, воспользуйтесь следующим:

```
[ window addSubview: myTableViewController.view ];
```

Чтобы протолкнуть его в контроллер навигации, воспользуйтесь следующим:

```
[ navigationController pushViewController: myTableViewController.view
        animated: YES ]
];
```

Улучшенный проводник файлов: TableDemo

Пример TableDemo из *главы 3* отображает простую таблицу, содержащую список файлов и папок, расположенных в домашней папке вашего приложения на iPhone. Данная версия этого примера создает отдельный раздел в алфавитном порядке для файлов и помещает каждый файл в раздел, соответствующий первой букве в его названии (рис. 10.4). Затем пример использует контроллер табличного представления и прикрепляет его к контроллеру панели навигации, чтобы предоставить пользователю кнопки редактирования и кнопку **Reload**. Пользователь сможет удалить элементы из списка нажатием кнопки **Edit** (не волнуйтесь, файлы на самом деле не будут удаляться). Также будет доступна функциональность "провести, чтобы удалить" (swipe), позволяющая пользователю проводить по ячейкам, чтобы удалить их.

Вы можете скомпилировать это приложение, показанное в листингах 10.6—10.10, с помощью SDK, создав проект приложения TableDemo на базе представления. Если вы хотите увидеть, как создавать все эти объекты с нуля, то удалите код Interface Builder.

Рис. 10.4. Пример TableDemo

Листинг 10.6. Прототипы делегата приложения TableDemo (TableDemoAppDelegate.h)

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

@class TableDemoViewController;

@interface TableDemoAppDelegate : NSObject <UIApplicationDelegate> {
    UIWindow *window;
    TableDemoViewController *viewController;
    UINavigationController *navigationController;
}

@property (nonatomic, retain) IBOutlet UIWindow *window;
```

```
@property (nonatomic, retain) IBOutlet TableDemoViewController
  *viewController;

@end
```

Листинг 10.7. Делегат приложения TableDemo (TableDemoAppDelegate.m)

```
#import "TableDemoAppDelegate.h"
#import "TableDemoViewController.h"

@implementation TableDemoAppDelegate

@synthesize window;
@synthesize viewController;

- (void)applicationDidFinishLaunching:(UIApplication *)application {
    CGRect screenBounds = [ [ UIScreen mainScreen ] bounds ];

    self.window = [ [ [ UIWindow alloc ] initWithFrame: screenBounds ]
      autorelease ];
    viewController = [ [ TableDemoViewController alloc ] init ];
    navigationController = [ [ UINavigationController alloc ]
        initWithRootViewController: viewController ];

    [ window addSubview: [ navigationController view ] ];
    [ window makeKeyAndVisible ];
}

- (void)dealloc {
    [ viewController release ];
    [ navigationController release ];
    [ window release ];
    [ super dealloc ];
}

@end
```

Листинг 10.8. Прототип контроллера представлений TableDemo (TableDemoViewController.h)

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

@interface TableDemoViewController : UITableViewController {
    int nActiveSections;
    NSMutableArray *fileList[27];
    NSMutableArray *activeSections;
    NSMutableArray *sectionTitles;
}

- (void) startEditing;
- (void) stopEditing;
- (void) reload;

@end
```

Листинг 10.9. Контроллер представлений TableDemo (TableDemoViewController.m)

```
#import "TableDemoViewController.h"

@implementation TableDemoViewController

- (id)init {
    self = [ super init ];

    if (self != nil) {
      /* Строим список файлов */
      [ self reload ];

    /* Инициализируем кнопки панели навигации */

    self.navigationItem.rightBarButtonItem =
       [ [ [ UIBarButtonItem alloc ]
       initWithBarButtonSystemItem: UIBarButtonSystemItemEdit
```

```
        target: self
        action: @selector(startEditing) ] autorelease ];

    self.navigationItem.leftBarButtonItem =
        [ [ [ UIBarButtonItem alloc ]
        initWithTitle:@"Reload"
        style: UIBarButtonItemStylePlain
        target: self
        action:@selector(reload) ]
       autorelease ];
    }

    return self;
}

- (NSArray *)sectionIndexTitlesForTableView:(UITableView *)tableView {

    return [ NSMutableArray arrayWithObjects:
         @"A", @"B", @"C", @"D", @"E", @"F",
         @"G", @"H", @"I", @"J", @"K", @"L",
         @"M", @"N", @"O", @"P", @"Q", @"R",
         @"S", @"T", @"U", @"V", @"W", @"X",
         @"Y", @"Z", @"#", nil
    ];
}

- (void) startEditing {
    [ self.tableView setEditing: YES animated: YES ];

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

```
- (void) stopEditing {
    [ self.tableView setEditing: NO animated: YES ];

    self.navigationItem.rightBarButtonItem =
       [ [ [ UIBarButtonItem alloc ]
       initWithBarButtonSystemItem: UIBarButtonSystemItemEdit
       target: self
       action: @selector(startEditing) ] autorelease ];
}

- (void) reload {
    NSDirectoryEnumerator *dirEnum;
    NSString *file;

    for (int i=0; i<27; i++) {
        fileList[i] = [ [ NSMutableArray alloc ] init ];
    }
    dirEnum = [ [ NSFileManager defaultManager ] enumeratorAtPath:
        NSHomeDirectory()
    ];

    while ((file = [ dirEnum nextObject ])) {
      char index = ([ file cStringUsingEncoding: NSASCIIStringEncoding ]
                   )[0];

      if (index >= 'a' && index <= 'z') {
          index -= 32;
      }
      if (index >= 'A' && index <= 'Z') {
          index -= 65;
          [ fileList[(int) index] addObject: file ];
      } else {
          [ fileList[26] addObject: file ];
      }
    }
```

```
    nActiveSections = 0;
    activeSections = [ [ NSMutableArray alloc ] init ];
    sectionTitles = [ [ NSMutableArray alloc ] init ];
    for (int i=0; i<27; i++) {
      if ([fileList[i] count ]>0) {
           nActiveSections++;
           [ activeSections addObject: fileList[i] ];
           if (i < 26)
              [ sectionTitles addObject: [ NSString stringWithFormat:
                   @"%c", i + 65 ] ];
           else
              [ sectionTitles addObject: @"0-9" ];
      }
    }

    [ self.tableView reloadData ];
}

- (NSString *)tableView:(UITableView *)tableView
    titleForHeaderInSection:(NSInteger)section
{
    return [ sectionTitles objectAtIndex: section ];
}

- (NSInteger)numberOfSectionsInTableView:(UITableView *)tableView {
    return nActiveSections;
}

- (NSInteger)tableView:(UITableView *)tableView
    numberOfRowsInSection:(NSInteger)section
{
    return [ [ activeSections objectAtIndex: section] count ];
}

- (NSInteger)tableView:(UITableView *)tableView sectionForSection
IndexTitle:(NSString *)title atIndex:(NSInteger) index {
```

```
    int i = 0;
    for (NSString * sectionTitle in sectionTitles) {
        if ([ sectionTitle isEqualToString: title ]) {
            [ tableView scrollToRowAtIndexPath:
                [ NSIndexPath indexPathForRow: 0 inSection: i ]
                   atScrollPosition: UITableViewScrollPositionTop
                     animated: YES ];
            return i;
        }
        i++;
    }
    return -1;
}

- (UITableViewCell *)tableView:(UITableView *)tableView
    cellForRowAtIndexPath:(NSIndexPath *)indexPath
{

    NSString *CellIdentifier = [[ activeSections objectAtIndex:
        [ indexPath indexAtPosition: 0 ]] objectAtIndex:
            [ indexPath indexAtPosition: 1 ] ];

    UITableViewCell *cell = [ tableView
        dequeueReusableCellWithIdentifier: CellIdentifier ];
    if (cell == nil) {
        cell = [ [ [ UITableViewCell alloc ]
            initWithFrame: CGRectZero reuseIdentifier: CellIdentifier
        ] autorelease ];

        cell.text = CellIdentifier;

        UIFont *font = [ UIFont fontWithName: @"Courier" size: 12.0 ];
        cell.font = font;
    }

    return cell;
}
```

```
- (void)tableView:(UITableView *)tableView
    commitEditingStyle:(UITableViewCellEditingStyle) editingStyle
    forRowAtIndexPath:(NSIndexPath *) indexPath
{
    if (editingStyle == UITableViewCellEditingStyleDelete) {

        /* Удаляем ячейку из источника данных */

        UITableViewCell *cell = [ self.tableView cellForRowAtIndexPath:
          indexPath ];
        for (int i = 0; i < [ [ activeSections objectAtIndex:
            [ indexPath indexAtPosition: 0 ] ] count ]; i++)
        {
           if ([ cell.text isEqualToString: [
                 [ activeSections objectAtIndex:
                   [ indexPath indexAtPosition: 0 ] ]
                objectAtIndex: i ] ])
           {
             [ [activeSections objectAtIndex:
               [indexPath indexAtPosition: 0]] removeObjectAtIndex: i];
           }
        }

        /* Удаляем ячейку из таблицы */

        NSMutableArray *array = [ [ NSMutableArray alloc ] init ];
        [ array addObject: indexPath ];
        [ self.tableView deleteRowsAtIndexPaths: array
            withRowAnimation: UITableViewRowAnimationTop ];
    }
}

- (void)tableView:(UITableView *)tableView
    didSelectRowAtIndexPath:(NSIndexPath *)indexPath
{
```

```
    UITableViewCell *cell = [ self.tableView cellForRowAtIndexPath:
      indexPath ];

    UIAlertView *alert = [ [ UIAlertView alloc ]
        initWithTitle: @"File Selected"
        message:
         [ NSString stringWithFormat: @"You selected the file '%@'",
             cell.text ]
        delegate: nil
        cancelButtonTitle: nil
        otherButtonTitles: @"OK", nil
    ];

    [ alert show ];
}

- (void)loadView {
    [ super loadView ];
}

- (BOOL)shouldAutorotateToInterfaceOrientation:
     (UIInterfaceOrientation)interfaceOrientation
{
    return YES;
}

- (void)didReceiveMemoryWarning {
    [ super didReceiveMemoryWarning ];
}

- (void)dealloc {
    for (int i=0; i<27; i++) {
     [ fileList[i] release ];
    }
    [ super dealloc ];
}

@end
```

Листинг 10.10. Функция `main` для TableDemo (main.m)

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

int main(int argc, char *argv[]) {
    NSAutoreleasePool * pool = [[NSAutoreleasePool alloc] init];
    int retVal = UIApplicationMain(argc, argv, nil,
                                   @"TableDemoAppDelegate");
    [pool release];
    return retVal;
}
```

Как это работает

Пример TableDemo работает следующим образом:

1. Приложение порождается точно так же, как и все предыдущие примеры, путем вызова программной функции `main`, которая вызывает метод `applicationDidFinishLaunching` класса `TableDemoAppDelegate` и создает соответствующее окно, контроллер представлений и классы контроллера навигации.
2. Контроллер представлений создается как экземпляр `UITableViewController`. Метод `init` контроллера подменяется для загрузки списка файлов в массив `fileList` объектов `NSMutableArray` исходя из первых букв имен этих файлов. Если первый символ — азбучный, то он добавляется в массивы `fileList` 0—25, в противном случае — в массив `fileList` 26.
3. Все 27 массивов `fileList` проверяются на предмет наличия содержимого. Непустые массивы добавляются в виде разделов в таблицу разделов посредством добавления их в массив `activeSections`. Заголовки массивов также хранятся по отдельности, поэтому мы можем отслеживать то, какие разделы отображаются.
4. Когда таблица визуализирована, автоматически вызываются методы источника данных контроллера. Метод `numberOfRowsInSection` возвращает количество строк в разделе, чье содержимое хранится в `activeSections`. `cellForRowAtIndexPath` создает новую ячейку, используя в качестве заголовка ячейки имя файла.
5. Если пользователь касается кнопки **Edit**, то уведомляется назначенный кнопке селектор `startEditing`. Это приводит к замене кнопки на кнопку

Done и разрешению редактирования таблицы. Таблица автоматически создает соответствующие отступы для ячеек и добавляет к ним значки удаления.

6. Если пользователь удалит какую-либо строку либо путем редактирования, либо методом "провести, чтобы удалить", у него будет запрошено подтверждение. После получения подтверждения запрос на удаление приведет к уведомлению о запросе метода `commitEditingStyle`. Этот метод проверит, является ли полученный запрос запросом на удаление, а затем удалит объект как из списка файлов, так и из таблицы.
7. Если пользователь нажмет кнопку **Reload**, то список файлов будет повторно прочтен, а таблица будет обновлена.

Для дальнейшего изучения

Посмотрим, можете ли вы взять что-либо из ранее изученного и применить к данному примеру.

- Используйте метод `fileAttributes` класса `NSDirectoryEnumerator` для определения того, какие файлы являются каталогми. Создайте по новому разделу для каждого каталога и добавьте содержимое каждого каталога в этот раздел. В качестве названия разделов используйте имена каталогов.
- Добавьте новый контроллер представлений, отображающий содержимое какого-либо файла в `UITextView`. Если этот файл является бинарным, то отобразите вывод в шестнадцатеричном формате. Когда пользователь коснется файла, то необходимо протолкнуть новый контроллер представлений в стек контроллера навигации, отображающего содержимое файла. Новый контроллер представлений должен иметь кнопку **Back**.
- Если вы этого еще не сделали, то в заголовочных файлах вашего SDK проверьте наличие прототипов UITableView.h, UITableViewCell.h и UITableViewController.h. Вы найдете их в папке /Developer/Platforms/iPhoneOS.platform внутри каталога Headers библиотеки UI Kit.

Индикаторы прогресса и активности

Индикаторы активности (activity indicators) сообщают пользователю о том, что некоторая операция находится в процессе выполнения. Индикаторы прогресса (progress indicators) делают то же самое, но еще дают пользователю

примерное представление о том, сколько еще осталось до завершения операции. Существуют два типа индикаторов, поддерживаемых SDK:

- `UIActivityIndicatorView`. Данный класс представляет собой вращающуюся анимацию наподобие часов. Такого типа анимацию вы можете видеть, к примеру, при включении поддержки Wi-Fi или Bluetooth на iPhone или же при загрузке вашего настольного компьютера Mac;
- `UIProgressView`. 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 представлений, листов предупреждений, табличных ячеек и любых других объектов, порождаемых от `UIView`.

Класс *UIActivityIndicatorView*: то, что вертится

Класс `UIActivityIndicatorView` является простым анимационным классом, достаточно небольшим, чтобы прикрепляться практически к 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 представления, к которому он прикрепляется:

```
UIActivityIndicatorView *activityIndicator =
  [[ UIActivityIndicatorView alloc ]
  initWithFrame: CGRectMake(260.0, 12.0, 25.0, 25.0)
];
```

Данный индикатор поддерживает три стиля, которые вы можете назначить путем задания свойства `activityIndicatorViewStyle` индикатора:

```
activityIndicator.activityIndicatorViewStyle =
  UIActivityIndicatorViewStyleGray;
```

Вы можете использовать следующие стили (табл. 10.3).

Таблица 10.3

Стиль	Описание
`UIActivityIndicatorViewStyleWhiteLarge`	Большой белый индикатор
`UIActivityIndicatorViewStyleWhite`	Белый индикатор стандартного размера
`UIActivityIndicatorViewStyleGray`	Серый индикатор для белых фонов

Чтобы автоматически скрыть представление, когда индикатор остановится, воспользуйтесь свойством `hidesWhenStopped`. По умолчанию индикатор скрывается, поэтому если вы хотите, чтобы индикатор отображался, даже когда он остановлен, то установите это свойство в `NO`:

```
activityIndicator.hidesWhenStopped = NO;
```

Объект индикатора прогресса может быть добавлен к любому существующему объекту представления, например, к ячейке таблицы или представлению:

```
[ tableCell addSubview: activityIndicator ];
```

Наконец, чтобы запустить или остановить анимацию, воспользуйтесь методами `startAnimation` и `stopAnimation`:

```
[ activityIndicator startAnimating ];
[ activityIndicator stopAnimating ];
```

UIProgressView: когда вращающиеся штучки не подходят

Объект `UIProgressView` является близким родственником объекту `UIActivityIndicatorView`. Вместо вращающейся анимации класс представления 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 она может более-менее точно отражать количество действительно выполненной приложением работы.

Чтобы создать представление прогресса, метод инициализации класса включает фрейм, определяющий размер строки и область отображения:

```
UIProgressView *progressView = [ [ UIProgressView alloc ]
    initWithFrame: CGRectMake(175.0, 20.0, 125.0, 25.0)
];
```

Данный индикатор поддерживает два различных стиля, которые вы можете назначить путем задания свойства `progressViewStyle` индикатора:

```
progressView.progressViewStyle = UIProgressViewStyleBar;
```

Вы можете использовать следующие стили (табл. 10.4).

Таблица 10.4

Стиль	**Описание**
`UIProgressViewStyleDefault`	Стандартная строка прогресса
`UIProgressViewStyleBar`	Темно-серая строка для использования на панелях инструментов

Чтобы отобразить представление прогресса, добавьте его к существующему объекту `UIView`. Вы можете добавить представления прогресса к ячейкам таблицы, панелям инструментов и другим классам представлений:

```
[ myToolbar addSubview: progressView ];
```

После того как строка прогресса отобразится на экране, ее прогресс может обновляться приложением, чтобы показать, где именно оно находится в процессе выполнения операции. Тип значений прогресса — число с плавающей точкой, а само значение находится в диапазоне между 0,0 и 1,0:

```
progressView.progress = 0.5;
```

Индикаторы сетевой активности

Если ваше приложение использует сеть, то оно должно предупреждать об этом пользователя посредством помещения сетевого индикатора в строку состояния iPhone. Для этого воспользуйтесь `UIApplication`-свойством `networkActivityIndicatorVisible`. Задайте это булево значение, чтобы разрешить или запретить сетевой индикатор:

```
UIApplication *myApp = [ UIApplication sharedApplication ];
myApp.networkActivityIndicatorVisible = YES;
```

Для дальнейшего изучения

- Воспользуйтесь своими знаниями контроллеров навигации, полученными в *главе 3*, для создания объекта `UIProgressView` на панели инструментов. Для заполнения строки используйте объект `NSTimer` или отдельный поток. Когда он исчерпает свой объем, обнулите представление прогресса. Это идеально подходит для приложений, которым требуется проверять в Интернете обновления или информацию о продуктах.
- Проверьте наличие прототипов UIProgressView.h и UIActivityIndicatorView.h. Вы найдете их в папке /Developer/Platforms/iPhoneOS.platform внутри каталога Headers библиотеки UI Kit.

Изображения

Библиотека UI Kit предоставляет классы для работы с отдельными изображениями и класс представления изображения, который может отображать их. Apple также предоставляет специальный контроллер навигации для выбора изображений из библиотеки.

Объект изображения

Класс `UIImage` инкапсулирует само изображение и относящиеся к нему данные. Он может использоваться для прорисовки напрямую внутри представления или выступать в роли контейнера объекта в более мощных классах представлений 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 объекты `UIImage` можно использовать в методе `drawRect` класса представления для прорисовки как изображений, так и узоров.

Инициализировать объект `UIImage` можно с содержимым файла, URL Web-узла, необработанными данными или же содержимым изображения Core Graphics. Существуют как статические методы, так и методы экземпляров; поэтому на изображения можно ссылаться, их можно кэшировать или порождать экземпляры объектов новых изображений в зависимости от потребностей вашего приложения.

Самый простой способ сослаться на изображение — это воспользоваться статическими методами класса `UIImage`. Вместо того чтобы управлять экземплярами изображений, статические методы предоставляют прямой интерфейс к общим объектам, расположенным во внутреннем кэше памяти библиотеки. Это помогает уменьшить беспорядок в вашем приложении и устраняет необходимость в проведении чистки. Для создания одних и тех же объектов существуют как статические методы, так и методы экземпляров.

Работа с файлами (статические методы)

Методы `imageNamed` и `imageWithContentsOfFile` позволяют вам получать доступ к файлам изображений либо по имени файла в вашем пакете, либо по полному пути к файлу соответственно. Чтобы получить доступ к файлу в программной папке вашего приложения, воспользуйтесь методом `imageNamed`:

```
UIImage *image = [ UIImage imageNamed: @"image.png" ];
```

Чтобы получить доступ к файлу, расположенному в любом другом месте в пределах вашей песочницы, воспользуйтесь методом `imageWithContentsOfFile`:

```
NSString *path = [ NSString stringWithFormat: @"%@/Documents/image.png",
                     NSHomeDirectory() ];
UIImage *image = [ UIImage imageWithContentsOfFile: path ];
```

Работа с URL и необработанными данными (статические методы)

Если изображение расположено в памяти резидентно, то вы можете инициализировать объект `UIImage` посредством создания объекта `NSData` и предоставления его в метод `imageWithData` как необработанного вводного параметра. В предыдущих разделах вы уже познакомились с классом `NSData`. Вы можете инициализировать структуры `NSData` различными способами, включая результаты выборки HTTP (HTTP fetch).

В приводимом далее примере предполагается, что переменная `imagePtr` будет указателем на необработанные данные изображения в вашем приложении, а переменная `imageSize` — размер изображения, хранящийся в памяти:

```
NSData *imageData = [ NSData initWithBytes: imagePtr length: imageSize ];
UIImage *image = [ UIImage imageWthData: imageData ];
```

Чтобы создать `UIImage`, хранящий содержимое Web-объекта, воспользуйтесь классом `NSData` для его загрузки, а затем инициализируйте ваш объект изображения:

```
NSURL *url = [ NSURL URLWithString:
    @" http://oreilly.com/catalog/covers/9781934356258_cat.gif"
];

NSData *imageData = [ NSData dataWithContentsOfUrl: url ];
UIImage *image = [ UIImage imageWithData: imageData ];
```

Работа с Core Graphics (статические методы)

Если вы программируете игры или другие графические приложения, используя библиотеку Core Graphics, то можете инициализировать объект `UIImage` с содержимым существующего объекта `CGImage`:

```
UIImage *image = [ UIImage imageWithCGImage: myCGImageRef ];
```

Работа с файлами (методы экземпляров)

Методы `initWithContentsOfFile` позволяют вам получать доступ к файлам изображений по пути (pathname). Вот соответствующий пример:

```
NSString *path = [ NSString stringWithFormat: @"%@/Documents/image.png",
    NSHomeDirectory()
];

UIImage *image = [ [ UIImage alloc ] initWithContentsOfFile: path ];
```

Работа с URL и необработанными данными (методы экземпляров)

Если изображение расположено в памяти резидентно, то вы можете инициализировать объект `UIImage` посредством создания объекта `NSData` и предоставления его в метод `imageWithData` как необработанного входного параметра.

В приводимом далее примере предполагается, что переменная `imagePtr` будет указателем на необработанные данные изображения в вашем приложении, а переменная `imageSize` — размер изображения, хранящегося в памяти:

```
NSData *imageData = [ NSData initWithBytes: imagePtr length: imageSize ];
UIImage *image = [ [ UIImage alloc ] initWithData: imageData ];
```

Чтобы породить UIImage, хранящий содержимое Web-объекта, воспользуйтесь классом NSData для его загрузки, а затем инициализируйте ваш объект изображения:

```
NSURL *url = [ NSURL URLWithString:
    @" http://oreilly.com/catalog/covers/9781934356258_cat.gif"
];

NSData *imageData = [ NSData dataWithContentsOfUrl: url ];
UIImage *image = [ [ UIImage alloc ] initWithData: imageData ];
```

Работа с Core Graphics (методы экземпляров)

Если вы программируете игры или другие графические приложения, используя библиотеку Core Graphics, то для инициализации объекта UIImage с содержимым имеющегося объекта CGImage существует и метод экземпляра:

```
UIImage *image = [ [ UIImage alloc ] initWithCGImage: myCGImageRef ];
```

Отображение изображения

Классы представлений используют встроенные процедуры прорисовки, которые вызываются, когда вызываются их методы drawRect. В отличие от других классов изображений, объект UIImage не может быть прикреплен напрямую к объекту представления как подчиненное представление. Вместо этого класс UIView вызывает метод drawInRect изображения из процедуры drawRect представления. Тем самым изображению приказывается визуализироваться в пределах области отображения класса UIView.

Метод drawRect объекта представления вызывается всякий раз, когда требуется визуализировать какую-либо часть его окна. Чтобы визуализировать содержимое UIImage внутри окна, вызовите метод drawInRect объекта:

```
- (void)drawRect:(CGRect)rect {
    CGRect myRect;

    myRect.origin.x = 0;
    myRect.origin.y = 0;
    myRect.size = myImage.size;

    [ myImage drawInRect: myRect ];
}
```

Будьте осторожны и не размещайте какие-либо новые объекты внутри метода drawRect, поскольку он вызывается каждый раз, когда необходимо перерисовать окно.

Метод drawRect вызывается только при первичной прорисовке представления. Чтобы вызвать обновление, воспользуйтесь методом setNeedsDisplay или методом setNeedsDisplayInRect класса представления:

```
[ myView setNeedsDisplay ];
[ myVIew setNeedsDisplayInRect: redrawThisRect ];
```

Вывод на экран узоров

Если изображение является узором, то для повторения этого изображения в пределах всей области отображения вы можете использовать другой метод, предоставляемый классом UIImage, — drawAsPatternInRect:

```
[ myImage drawAsPatternInRect: rect ];
```

Этот метод повторяет указанное изображение в пределах отображаемого фрейма.

Ориентация

Ориентация (orientation) изображения определяет, как оно повернуто относительно экрана. Поскольку iPhone можно располагать шестью различными способами, то может возникнуть необходимость при изменении ориентации устройства поворачивать и все ваши изображения. Чтобы задать ориентацию, воспользуйтесь свойством imageOrientation изображения:

```
myImage.imageOrientation = UIImageOrientationUp;
```

Вы можете задать одну из следующих ориентаций (табл. 10.5).

Таблица 10.5

Ориентация	Описание
UIImageOrientationUp	Заданная по умолчанию ориентация
UIImageOrientationDown	Изображение повернуто на 180°
UIImageOrientationLeft	Изображение повернуто на 90° против часовой стрелки

Таблица 10.5 *(окончание)*

Ориентация	Описание
`UIImageOrientationRight`	Изображение повернуто на 90° по часовой стрелке
`UIImageOrientationUpMirrored`	Вверх, зеркально отображено по горизонтали
`UIImageOrientationDownMirrored`	Вниз, зеркально отображено по горизонтали
`UIImageOrientationLeftMirrored`	Изображение повернуто на 90° против часовой стрелки, зеркально отображено по вертикали
`UIImageOrientationRightMirrored`	Изображение повернуто на 90° по часовой стрелке, зеркально отображено по вертикали

Размер изображения

Считать размер изображения можно с помощью свойства `size`. Оно предоставляет структуру `CGSize`, содержащую переменные `width` и `height`:

```
CGSize imageSize = myImage.size;
NSLog(@"Image size: %dx%d", imageSize.width, imageSize.height);
```

Развлечение с изображениями и узорами: ImageFun

Данный пример иллюстрирует вывод на экран изображений и узоров в рамках метода `drawRect` класса представления. Вы создадите пустой подкласс `UIView`, а затем подмените метод `drawRect` для включения процедур прорисовки, отображая в основном окне узор и значок. Этот пример будет загружать два файла изображений, которые он будет использовать для прорисовки узора и изображения на экране (рис. 10.5). Изображение будет постепенно проявляться по таймеру.

Вы можете скомпилировать это приложение, приведенное в листингах 10.11—10.13, с помощью SDK, создав проект оконного приложения ImageFun. Если вы хотите увидеть, как создавать все эти объекты с нуля, то удалите код Interface Builder.

Рис. 10.5. Пример ImageFun

Листинг 10.11. Прототипы делегата приложения ImageFun (ImageFunAppDelegate.h)

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

@interface ImageFunView : UIView
{
    UIImage *pattern;
    UIImage *image;
    float alpha;
}
- (void)drawRect:(CGRect)rect;
@end

@interface ImageFunAppDelegate : NSObject <UIApplicationDelegate> {
    UIWindow *window;
    ImageFunView *mainView;
}

@property (nonatomic, retain) IBOutlet UIWindow *window;

@end
```

Листинг 10.12. Делегат приложения ImageFun (ImageFunAppDelegate.m)

```
#import "ImageFunAppDelegate.h"

@implementation ImageFunView

- (id)initWithFrame:(CGRect) rect {
    self = [ super initWithFrame: rect ];
    if (self != nil) {
        NSLog(@"Loading pattern");
        NSURL *url = [ NSURL URLWithString:
            @"http://www.zdziarski.com/demo/1.png" ];
        pattern = [ [ UIImage alloc ] initWithData:
            [ NSData dataWithContentsOfURL: url ]
        ];

        NSLog(@"Loading image");
        NSURL *url2 = [ NSURL URLWithString:
            @"http://www.zdziarski.com/demo/2.png" ];
        image = [ [ UIImage alloc ] initWithData:
            [ NSData dataWithContentsOfURL: url2 ]
        ];

        alpha = 0.0;
    }
    return self;
}

- (void)drawRect:(CGRect)rect {
    CGRect myRect;

    myRect.size = image.size;
    myRect.origin.x = (320.0 - image.size.width) / 2;
    myRect.origin.y = (460.0 - image.size.height) / 2;

    [ pattern drawAsPatternInRect: rect ];
```

```
    [ image drawInRect: myRect blendMode: kCGBlendModeNormal
        alpha: alpha];

    NSTimer *timer = [ NSTimer scheduledTimerWithTimeInterval: 0.01
        target: self
      selector: @selector(handleTimer:)
       serInfo: nil
       repeats: NO
    ];
}

- (void) handleTimer: (NSTimer *) timer {
    if (alpha < 1.0) {
        alpha += 0.01;
        [ self setNeedsDisplay ];
    }
}

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

@implementation ImageFunAppDelegate

@synthesize window;

- (void)applicationDidFinishLaunching:(UIApplication *)application {
    CGRect screenBounds = [ [ UIScreen mainScreen ] applicationFrame ];
    CGRect viewRect = screenBounds;
    viewRect.origin.x = viewRect.origin.y = 0;

    self.window = [ [ [ UIWindow alloc ] initWithFrame: screenBounds ]
        autorelease
    ];

    mainView = [ [ ImageFunView alloc ] initWithFrame: viewRect ];
```

```
    [ window addSubview: mainView ];
    [ window makeKeyAndVisible ];
}

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

@end
```

Листинг 10.13. Функция `main` для ImageFun (main.m)

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

int main(int argc, char *argv[]) {
    NSAutoreleasePool * pool = [[NSAutoreleasePool alloc] init];
    int retVal = UIApplicationMain(argc, argv, nil,
                                   @"ImageFunAppDelegate");
    [pool release];
    return retVal;
}
```

Изображение с представлением: *UIIMageView*

Класс `UIIMageView` предоставляет способ для инкапсуляции изображения в классе представления или для предоставления анимации в рамках представления. Это оказывается полезным, когда изображение необходимо прикрепить к другим объектам представлений, или при разработке таких приложений, как презентации слайдов, в которых вся область представления может содержать изображение.

Объект `UIImageView` выступает для `UIImage` в роли обертки класса представления, т. е. сначала вы создаете объект `UIImage`, а затем вы используете его

для инициализации объекта `UIImageView` с помощью метода `initWithImage` или метода `setImage` класса `UIImageView`:

```
UIImage *image = [ UIImage imageNamed: @"image.png" ];
UIImageView *imageView = [ [ UIImageView alloc ]
    initWithImage: image
];
```

Чтобы построить анимацию, вы можете присвоить массив изображений представлению изображения:

```
NSMutableArray *myImages = [ [ NSMutableArray alloc ] init ];
[ myImages addItem: myImage1 ];
[ myImages addItem: myImage2 ];
[ myImages addItem: myImage3 ];
imageView.animationImages = myImages;
```

После описания анимации задайте ее свойства, определяющие длительность и количество повторов:

```
imageView.animationDuration = 5.0; /* Пять секунд */
imageView.repeatCount = 3;
```

После настройки объекта `UIImageView` вы можете прикрепить его к объекту представления любого типа, ячейке таблицы или другому подобному объекту:

```
[ myOtherView addSubview: imageView ];
```

Чтобы масштабировать изображение, создайте новый фрейм, соответствующий нужному размеру. Затем вы сможете применить новый размер изображений, присвоив фрейм свойству `frame` объекта:

```
CGRect rect = imageView.frame;
CGSize size = CGSizeMake(160.0, 240.0);
rect.size = size;
imageView.frame = rect;
```

Чтобы разрешить или запретить анимацию представления, воспользуйтесь методами `startAnimating` и `stopAnimating`:

```
[ imageView startAnimating ];
if ([ imageView isAnimating ] == YES) {
    [ imageView stopAnimating ];
}
```

В *главах 12* и *13* вы будете активно использовать класс `UIImageView`.

Выборщики изображений

Выборщик изображений (image picker) — это тип класса контроллера навигации, позволяющий вам добавлять в ваше приложение простые селекторы изображений или интерфейс камеры. Пользователю предоставляется экран выбора изображений, на котором он может выбрать фотографию из своей библиотеки фотографий, сохраненного фотоальбома или непосредственно с камеры. Когда пользователь выбирает фотографию, уведомляется делегат выборщика с помощью методов из протокола `UIImagePickerDelegate`.

Выборщик изображений создается как объект `UIImagePickerController` и может быть добавлен к окну как самостоятельный контроллер навигации:

```
UIImagePickerController *picker = [ [ UIImagePickerController alloc ]
                                    init ];
[ window addSubview: [ picker view ] ];
```

Источники изображений

Для предоставления пользователю с помощью свойства `sourceType` можно определить различные источники изображений:

```
picker.sourceType = UIImagePickerControllerSourceTypePhotoLibrary;
```

Вы можете использовать следующие источники (табл. 10.6).

Таблица 10.6

Стиль	Описание
`UIImagePickerControllerSourceTypePhotoLibrary`	Библиотека фотографий
`UIImagePickerControllerSourceTypeCamera`	Камера
`UIImagePickerControllerSourceTypeSavedPhotos-Album`	Сохраненные фотографии

Редактирование изображений

Чтобы разрешить пользователю перемещать и масштабировать изображение по своему вкусу, разрешите редактирование изображений, установив свойство `allowsImageEditing` в `YES`:

```
picker.allowsImageEditing = YES;
```

Выбор изображений

Когда пользователь выбирает картинку, посредством метода `didFinishPickingImage` уведомляется делегат выборщика. Делегату предоставляется объект `UIImage`, содержащий все свойства редактирования, если оно разрешено.

Чтобы назначить делегата выборщику, задайте свойство делегата выборщика:

```
picker.delegate = self;
```

Чтобы получать уведомления, когда пользователь выбирает изображение, добавьте к вашему классу-делегату следующий метод:

```
- (void)imagePickerController:(UIImagePickerController *)picker
    didFinishPickingImage:(UIImage *)image
    editingInfo:(NSDictionary *)editingInfo
{
    /* Код для обработки выбора изображения */
}
```

Параметры предоставляют вам указатель на контроллер выборщика изображений, уведомляющего о действии, поэтому в одном делегате вы можете обрабатывать несколько выборщиков. Также вам предоставляется указатель на сам объект `UIImage` и объект словаря, содержащий информацию о том, как изображение было масштабировано и перемещалось на экране.

Также вы захотите получать уведомления в том случае, если пользователь отменит выбор изображений. Для этого добавьте в ваш делегат метод `imagePickerControllerDidCancel`. Он будет вызываться с указателем на того выборщика изображений, который был отменен:

```
- (void)imagePickerControllerDidCancel:(UIImagePickerController *)picker
{
    /* Код для обработки отмены выбора изображений */
}
```

Свойства клавиатуры

Представляя iPhone в одном из своих самых ожидаемых докладов, Стив Джобс обрисовал устройство, предлагающее совершенно новый подход к эргономике интерфейса. Этот подход не ограничивается созданием кнопок в

области приложения, а позволяет создавать абсолютно новый тип интерфейса пользователя, отвечающий специфическим особенностям приложений. Джобс ненавидит физические кнопки, явно присутствующие на клавиатурах, поскольку Apple разработала почти дюжину различных стилей "виртуальных" клавиатур для iPhone и предоставила элегантный интерфейс для их определения исходя из того, какого рода требуется ввод.

Когда пользователь касается текстового поля, то среда исполнения автоматически вызывает клавиатуру для осуществления пользователем ввода. Это позволяет разработчику заботиться только об отображении текстовых полей и не думать о проблемах управления клавиатурой. Apple изящно спроектировала свою библиотеку, поэтому текстовое поле имеет полную власть над действиями самой клавиатуры, а не над объектом клавиатуры. Когда клавиатура появляется, то она автоматически настраивает себя в соответствии с тем поведением, которое было задано в текстовом поле.

Рассматривайте такой подход как водительские параметры в машине. Каждый водитель предпочитает собственные настройки автомобиля: высоту сидений, наклоны зеркал и расположение педалей. Если автомобиль захочет вести кто-либо другой, то вы не будете думать о том, как установить новое водительское сидение только из-за смены водителей. Вместо этого водительское сидение в автомобиле просто будет подогнано под нового водителя. Считайте текстовое поле водителем, а клавиатуру — водительским сиденьем, параметры которого настраиваются.

Для таблиц настроек и других подобных представлений, состоящих из множества различных текстовых полей, вы можете определить различные поведения клавиатур, чтобы каждая ячейка имела бы собственный стиль. Все описываемые в этом разделе свойства доступны в большинстве текстовых объектов, т. е. в тех, которые импортируют `UITextInputTraits`. Их вы можете найти в прототипе UITextInputTraits.h.

Стиль клавиатуры

Библиотека UI Kit поддерживает восемь различных стилей клавиатуры. Для каждого текстового поля можно определить свой стиль.

Задайте стиль с помощью метода `keyboardType` текстового объекта:

```
textView.keyboardType = UIKeyboardTypePhonePad;
```

Поддерживаются следующие стили клавиатур (табл. 10.7).

Таблица 10.7

Стиль	**Описание**
`UIKeyboardTypeDefault`	Клавиатура по умолчанию, присутствуют все символы
`UIKeyboardTypeASCIICapable`	Клавиатура по умолчанию с поддержкой ASCII
`UIKeyboardTypeNumbersAnd-Punctuation`	Стандартная телефонная клавиатура, поддерживающая символы: + * #
`UIKeyboardTypeURL`	URL-клавиатура с кнопкой **.COM**, поддерживает только URI-символы
`UIKeyboardTypeNumberPad`	Цифровая клавиатура для ввода чисел
`UIKeyboardTypePhonePad`	Телефонная клавиатура для ввода номеров телефонов
`UIKeyboardTypeNamePhonePad`	Телефонная клавиатура, поддерживающая еще и ввод имен
`UIKeyboardTypeEmailAddress`	Клавиатура для ввода адресов электронной почты

Раскладка как обычной клавиатуры, так и телефонной имеет одинаковый размер, поэтому для переключения между ними не требуется никаких дополнительных изменений окна.

Внешний вид клавиатуры

Помимо типа клавиатуры вы можете настроить и ее внешний вид, задав свойство `keyboardAppearance`:

```
textView.keyboardAppearance = UIKeyboardAppearanceDefault;
```

Существуют следующие виды клавиатур (табл. 10.8).

Таблица 10.8

Стиль	**Описание**
`UIKeyboardAppearanceDefault`	Внешний вид по умолчанию, светло-серый
`UIKeyboardAppearanceAlert`	Темно-серый/графитовый

Клавиша возврата

Для клавиатур, имеющих клавишу возврата, с помощью свойства `returnKeyType` текстового объекта вы можете назначить различные стили этой кнопки:

```
textView.returnKeyType = UIReturnKeyGo;
```

Поддерживаются следующие стили (табл. 10.9).

Таблица 10.9

Стиль	**Описание**
`UIReturnKeyDefault`	По умолчанию: серая кнопка с меткой **Return**
`UIReturnKeyGo`	Синяя кнопка с меткой **Go**
`UIReturnKeyGoogle`	Синяя кнопка с меткой **Google**, используемой для осуществления поиска
`UIReturnKeyJoin`	Синяя кнопка с меткой **Join**
`UIReturnKeyNext`	Серая кнопка с меткой **Next**
`UIReturnKeyRoute`	Синяя кнопка с меткой **Route**
`UIReturnKeySearch`	Синяя кнопка с меткой **Search**
`UIReturnKeySend`	Синяя кнопка с меткой **Send**
`UIReturnKeyYahoo`	Синяя кнопка с меткой **Yahoo!**, используемая для осуществления поиска
`UIReturnKeyDone`	Синяя кнопка с меткой **Done**
`UIReturnKeyEmergencyCall`	Кнопка экстренного вызова

Автоматическое выделение прописными буквами

Клавиатуры могут автоматически делать первую букву новой строки или предложения прописной. Для включения этой возможности задайте свойство `autocapitalizationType` текстового объекта:

```
textView.autocapitalizationType = UITextAutocapitalizationTypeNone;
```

Поддерживаются следующие типы автоматического выделения прописными буквами:

- `UITextAutocapitalizationTypeNone`;
- `UITextAutocapitalizationTypeWords`;
- `UITextAutocapitalizationTypeSentences`;
- `UITextAutocapitalizationTypeAllCharacters`.

Автозамена

При вводе текста объекты текстового представления и клавиатуры совместно предоставляют возможные замены для ошибок набора. Эта функциональность использует встроенный словарь самых распространенных неправильно набираемых слов, а также кэш ввода, который отслеживает ваш собственный ввод символов. iPhone хранит словарь в /private/var/mobile/Library/Keyboard/dynamic-text.dat.

По умолчанию автозамена разрешена, но вы можете отключить ее с помощью свойства `autocorrectionType`:

```
textView.autocorrectionType = UITextAutocorrectionTypeDefault ;
```

Существуют следующие типы автозамены:

- `UITextAutocorrectionTypeDefault`;
- `UITextAutocorrectionTypeNo`;
- `UITextAutocorrectionTypeYes`.

Защищенный ввод текста

При вводе паролей или других персональных данных в текстовое поле вводимая информация не должна кэшироваться в iPhone. Включение возможности защищенного ввода текста запрещает функции автозамены и кэширования для данного текстового поля. Чтобы активировать режим защищенного ввода текста, задайте свойство `secureTextEntry`:

```
textView.secureTextEntry = YES;
```

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 осуществления выбора. Класс `UIPickerView` был создан скорее как полноценный класс представления, нежели элемент управления, из-за его сложности и размера, занимаемого им на экране. Это позволяет прикреплять выборщиков к другим представлениям или окнам.

Создание выборщика

Класс `UIPickerView` занимает на экране огромное пространство размером в 320×216 пикселов, но может быть смещен по вертикали в любом месте окна. Как и таблицы, класс `UIPickerView` использует источник данных. Но в отличие от таблиц, выборщики не используют индексные пути, а ссылаются на каждую строку посредством значения `NSInteger`. Выборщики могут иметь несколько циферблатов, каждый их которых называется *компонентом* (component).

Представление выборщика использует в качестве источника данных делегата, позволяя тем самым источнику данных выводиться в отдельный класс или контроллер представлений, как в случае с таблицами в *главе 3*, а также таблицами настроек и списками разделов, описанными ранее в этой главе.

Область отображения представления выборщика задается автоматически во фрейме по умолчанию размером в 320×216 пикселов. Если вы попытаетесь инициализировать выборщик собственным размером фрейма, то он будет проигнорирован. Вы можете разместить выборщик в любом месте экрана, однако, как правило, они располагаются сверху или снизу:

```
UIPickerView *pickerView = [ [ UIPickerView alloc ]
    initWithFrame: CGRectMake(0.0, 280.0, 0.0, 0.0)];
pickerView.delegate = self;
pickerView.dataSource = self;
```

Получение свойств выборщика

Многие из свойств представления выборщика в SDK были сделаны частными, тем самым уменьшая имеющийся у разработчика контроль над ними по

сравнению с имеющимся контролем у встроенных классов. Выборщики имеют различные свойства для управления звуком, осуществления множественного выбора и других эстетических параметров, однако все эти интерфейсы недоступны разработчику SDK. Единственным эстетическим параметром, которым может управлять разработчик, является окно выбора.

Чтобы для текущего выбранного варианта отобразить полупрозрачное окно, установите свойство `showsSelectionIndicator` в `YES`:

```
pickerView.showsSelectionIndicator = YES;
```

Источник данных выборщика

После создания представления выборщика вы должны запрограммировать источник данных, предоставляющий информацию о структуре выборщика. Для создания источника данных необходимы приводимые далее методы. Это обязательные компоненты протокола `UIPickerViewDataSource`:

- `numberOfComponentsInPickerView` определяет количество отдельных щелчков колесика (столбцов), отображаемых внутри представления выборщика;
- `numberOfRowsInComponent`. Каждому циферблату в выборщике вы можете назначить свое количество возможных значений (строк). Данный метод должен возвращать общее количество строк для указанного номера циферблата. Помимо этого, протокол `UIPickerViewDelegate` реализует следующие методы для получения информации о компонентах выборщика;
- `titleForRow` возвращает действительное значение циферблата для заданной строки заданного циферблата (компонента). Данное значение возвращается как объекты `NSString`;
- `viewForRow` может подменять заданное по умолчанию поведение выборщика для отображения любого класса на циферблатах компонента;
- `widthForComponent` возвращает ширину заданного компонента (циферблата). Если этот метод не реализован, то выборщик будет автоматически устанавливать размер по ширине компонента;
- `rowHeightForComponent` возвращает высоту заданного компонента (циферблата). Если этот метод не реализован, то выборщик будет автоматически устанавливать размер по высоте компонента.

Прототипы и функция для этих методов будут продемонстрированы в примере, приводимом далее в этом разделе.

Отображение выборщика

Создав и сконфигурировав представление выборщика, а также написав методы источника данных, вы подготовились к прикреплению выборщика к вашему контроллеру представлений:

```
[ self.view addSubview: pickerView ];
```

Считывание выборщика

Самый прямой способ для получения порядкового номера выделенного столбца в представлении выборщика — это воспользоваться методом `selectedRowInComponent` представления:

```
int selectedRow = [ pickerView selectedRowInComponent: 0 ];
```

Также существует метод-делегат, который уведомляется всякий раз, когда пользователь выделяет какую-либо строку в представлении выборщика. Воспользуйтесь этим для предупреждения объекта, чтобы он мог ответить на новое выделение строки:

```
pickerView.delegate = myObject;
```

Чтобы получать уведомление, когда меняется значение циферблата, добавьте в ваш класс следующий метод-делегат `didSelectRow`:

```
- (void)pickerView:(UIPickerView *)pickerView
   didSelectRow:(NSInteger)row inComponent:(NSInteger)component
{
    NSLog(@"Selected row %d from dial %d", row, component);

    /* Код для обработки выделения строки */
}
```

Выбор типа вашего носа: NosePicker

В этом примере пользователю предоставляется `UIPickerView`, содержащий различные типы и размеры носов (рис. 10.6). Сначала создается контроллер представления, который затем принимает в качестве подчиненного представления представление выборщика. Вы сможете прокручивать список носов и выбирать из них один.

Рис. 10.6. Пример NosePicker

Вы можете скомпилировать это приложение, приведенное в листингах 10.14—10.18, с помощью SDK, создав проект приложения NosePicker на базе представления. Если вы хотите увидеть, как создавать все эти объекты с нуля, то удалите код Interface Builder.

Листинг 10.14. Прототипы делегата приложения NosePicker (NosePickerAppDelegate.h)

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

@class NosePickerViewController;

@interface NosePickerAppDelegate : NSObject <UIApplicationDelegate> {
    UIWindow *window;
    NosePickerViewController *viewController;
}
```

```
@property (nonatomic, retain) IBOutlet UIWindow *window;
@property (nonatomic, retain) IBOutlet NosePickerViewController
   *viewController;

@end
```

Листинг 10.15. Делегат приложения NosePicker (NosePickerAppDelegate.m)

```
#import "NosePickerAppDelegate.h"
#import "NosePickerViewController.h"

@implementation NosePickerAppDelegate

@synthesize window;
@synthesize viewController;

- (void)applicationDidFinishLaunching:(UIApplication *)application {
    CGRect screenBounds = [ [ UIScreen mainScreen ] bounds ];

    self.window = [ [ [ UIWindow alloc ] initWithFrame: screenBounds ]
      autorelease ];
    viewController = [ [ NosePickerViewController alloc ] init ];

    [ window addSubview: viewController.view ];
    [ window makeKeyAndVisible ];
}

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

@end
```

Листинг 10.16. Прототипы контроллера представления NosePicker (NosePickerViewController.h)

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

@protocol UIPickerViewDataSource, UIPickerViewDelegate;

@interface NosePickerViewController : UIViewController
   <UIPickerViewDelegate,
    UIPickerViewDataSource> {
    UIPickerView *pickerView;
    UITextView *textView;
    NSMutableArray *noses;
    NSMutableArray *sizes;
    int selection[2];
}
- (void)updateText;
- (NSInteger)numberOfComponentsInPickerView:(UIPickerView *)pickerView;
- (NSInteger)pickerView:(UIPickerView *)pickerView
    numberOfRowsInComponent:(NSInteger)component;
- (NSString *)pickerView:(UIPickerView *)pickerView
    titleForRow:(NSInteger)row forComponent:(NSInteger)component;
- (void)pickerView:(UIPickerView *)pickerView
    didSelectRow:(NSInteger)row inComponent:(NSInteger)component;

@end
```

Листинг 10.17. Контроллер представлений NosePicker (NosePickerViewController.m)

```
#import "NosePickerViewController.h"

@implementation NosePickerViewController

- (id)init {
    self = [ super init ];
    if (self != nil) {
```

```
        noses = [ [ NSMutableArray alloc ] init ];
        [ noses addObject: @"Straight" ];
        [ noses addObject: @"Aquiline" ];
        [ noses addObject: @"Retrousse" ];
        [ noses addObject: @"Busque" ];
        [ noses addObject: @"Sinuous" ];
        [ noses addObject: @"Melanesian" ];
        [ noses addObject: @"African" ];

        sizes = [ [ NSMutableArray alloc ] init ];
        [ sizes addObject: @"Small" ];
        [ sizes addObject: @"Medium" ];
        [ sizes addObject: @"Large" ];
        [ sizes addObject: @"Super-Size" ];

        selection[0] = selection[1] = 0;
    }
    return self;
}

- (void)loadView {
    CGRect bounds = [ [ UIScreen mainScreen ] applicationFrame ];

    [ super loadView ];

    pickerView = [ [ UIPickerView alloc ]
     initWithFrame: CGRectMake(0.0, bounds.size.height - 216.0, 0.0, 0.0)
    ];

    pickerView.delegate = self;
    pickerView.dataSource = self;
    pickerView.showsSelectionIndicator = YES;
    [ self.view addSubview: pickerView ];

    textView = [ [ UITextView alloc ] initWithFrame:
     CGRectMake(0.0, 0.0, bounds.size.width, bounds.size.height - 216.0)
    ];
```

```
    textView.font = [ UIFont fontWithName: @"Arial" size: 18.0 ];
    textView.textAlignment = UITextAlignmentCenter;
    textView.editable = NO;

    [ self updateText ];
    [ self.view addSubview: textView ];
}

- (void)updateText {
    textView.text = [ NSString stringWithFormat:
        @"One %@, %@ schnoz:\nComing right up!\n",
        [ sizes objectAtIndex: selection[1] ],
        [ noses objectAtIndex: selection[0] ]
    ];
}

- (NSInteger)numberOfComponentsInPickerView:(UIPickerView *)pickerView {
    return 2;
}

- (NSInteger)pickerView:(UIPickerView *)pickerView
    numberOfRowsInComponent:(NSInteger)component
{
    switch(component) {
        case(0):
            return [ noses count ];
            break;
        case(1):
            return [ sizes count ];
            break;
    }
    return 0;
}

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

```
    switch (component) {
        case(0):
            return [ noses objectAtIndex: row ];
            break;
        case(1):
            return [ sizes objectAtIndex: row ];
    }
    return nil;
}

- (void)pickerView:(UIPickerView *)pickerView
    didSelectRow:(NSInteger)row inComponent:(NSInteger)component
{
    selection[component] = row;
    [ self updateText ];
}

- (BOOL)shouldAutorotateToInterfaceOrientation:
    (UIInterfaceOrientation)interfaceOrientation
{
    return (interfaceOrientation == UIInterfaceOrientationPortrait);
}

- (void)didReceiveMemoryWarning {
    [ super didReceiveMemoryWarning ];
}

- (void)dealloc {
    [ pickerView release ];
    [ noses release ];
    [ sizes release ];
    [ super dealloc ];
}

@end
```

Листинг 10.18. Функция `main` для NosePicker (main.m)

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

int main(int argc, char *argv[]) {
    NSAutoreleasePool * pool = [[NSAutoreleasePool alloc] init];
    int retVal = UIApplicationMain(argc, argv, nil,
                                   @"NosePickerAppDelegate");
    [pool release];
    return retVal;
}
```

Как это работает

1. При порождении приложения создается новый контроллер представления, который, в свою очередь, инициализирует массив типов носов и их размеров. Также инициализируется переменное хранилище для выбранных в текущий момент циферблатов, хранящееся в `selection`.
2. Когда вызывается метод `loadView` контроллера представлений, создаются объекты `UIPickerView` и `UITextView`, которые добавляются к контроллеру представлений в качестве подчиненных представлений. Контроллеру представлений назначаются делегат и источник данных для представления выборщика. Метод `updateText` контроллера вызывается единожды для задания текста в текстовом представлении.
3. Внутренняя структура класса `UIPickerView` вызывает свой источник данных, который возвращает количество компонентов, строк и заголовков для каждой строки в представлении выборщика.
4. Когда пользователь выделяет новый элемент на любом циферблате представления выборщика, уведомляется делегат представления (контроллер представления) посредством метода `didSelectRow` представления выборщика. Он обновляет последнюю выделенную строку для заданного циферблата и отображение текста, чтобы показать выделенную в данный момент комбинацию типа и размера носа.

Для дальнейшего изучения

Когда вы покончите с выбором вашего носа, выполните следующие задания.

- Проверьте наличие прототипа UIPickerView.h. Вы найдете его в папке /Developer/Platforms/iPhoneOS.platform внутри каталога Headers библиотеки UI Kit.
- Внутри прототипа вы обнаружите альтернативное множество методов источника данных, которые вместо объектов `NSString` возвращают объекты `UIView`. Воспользуйтесь ими для создания собственных объектов `UIImageView` для выборщика.

Выборщик даты и времени

Класс `UIDatePicker` является элементом управления, который инкапсулирует пользовательский класс `UIPickerView` специально для приема вводимых даты, времени и длительности. Выборщик даты автоматически настраивает свои столбцы в соответствии с указанным стилем, поэтому при конфигурировании циферблатов не нужно выполнять никакой работы на низком уровне. Кроме того, он может быть настроен на любой диапазон дат.

`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(0.0, 0.0, 0.0, 0.0)];
```

По умолчанию выборщик представляет текущие дату и время, а также циферблаты для выбора дня недели, месяца и дня, часа, минуты и времени суток (AM/PM). По умолчанию пользователь может выбрать любую комбинацию даты и времени. Дальнейшие настройки операций выборщика рассмотрены в следующих подразделах.

Режимы *DatePicker*

Выборщик дат и времени поддерживает четыре различных режима выбора. Режим задается с помощью свойства `DatePickerMode` следующим образом:

```
datePicker datePickerMode = UIDatePickerModeTime;
```

Поддерживаются следующие режимы (табл. 10.10).

Таблица 10.10

Режим	**Описание**
`UIDatePickerModeTime`	Выбор часа, минуты и времени суток (A.M./P.M.)
`UIDatePickerModeDate`	Выбор месяца, дня и года
`UIDatePickerModeDateAndTime`	По умолчанию; выбор недели + + месяца + дня, часа, минуты и времени суток (A.M./P.M.)
`UIDatePickerModeCountDownTimer`	Выбор часа и минуты для отображения в таймерах

Временные интервалы

Минутный циферблат может быть настроен на отображение минут в различных интервалах, если этот интервал нацело делится на 60. По умолчанию отображается минутный циферблат с одноминутными интервалами. Чтобы выбрать другой интервал, установите свойство `minuteInterval` в нужный интервал:

```
datePicker.minuteInterval = 10;
```

Диапазоны дат

Диапазон разрешенных дат можно задать с помощью свойств `minimumDate` и `maximumDate`. Если пользователь попытается прокрутить 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.minimumDate = minDate;
datePicker.maximumDate = maxDate;
```

Если не установлены одно или оба этих свойства, то по умолчанию пользователю разрешается выбирать любую дату в прошлом или будущем. Это может оказаться полезным, к примеру, при выборе даты рождения, которая может оказаться любой датой из прошлого вплоть до сегодняшнего дня.

Чтобы задать дату, которую вы хотите отображать по умолчанию, установите свойство `date`:

```
datePicker.date = minDate;
```

Иначе вы можете воспользоваться методом `setDate`. Если вы решите анимировать выбор даты, то циферблаты будут прокручиваться до указанной вами даты:

```
[ datePicker setDate: maxDate animated: YES ];
```

Отображение выборщика даты

После создания выборщика даты его можно прикрепить к объекту представления с помощью того же метода, что и для `UIPickerView`:

```
[ self addSubview: datePicker ];
```

По умолчанию высота выборщика составляет 216 пикселов, 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 представлению, например, при выходе из таблицы настроек.

При считывании свойство `date` предоставляет объект `NSDate`:

```
NSDate *selectedDate = datePicker.date;
```

Поскольку выборщик дат является подклассом класса `UIControl` (в отличие от `UIPickerView`), то вы также можете подключить в структуру уведомлений класса `UIControl` делегата:

```
[ datePicker addTarget: self action:
    @selector(dateChanged:)
    forControlEvents:UIControlEventValueChanged
];
```

Ваш класс действий будет вызываться всякий раз, когда пользователь будет выбирать новую дату:

```
- (void) dateChanged: (id)sender
{
    UIDatePicker *control = (UIDatePicker *) sender;
    NSDate *selectedDate = control.date;

  /* Код для реакции на изменения даты */
}
```

Пример: выборщик Дня независимости

Данный пример иллюстрирует использование основного объекта выборщика даты для осуществления выбора даты в диапазоне между датой Бостонского чаепития (Boston Tea Party — December 16, 1773) и Днем независимости Америки (American Independence Day — July 4, 1776). Пример просто создает объект `UIDatePicker` и отображает его пользователю (рис. 10.7). Когда пользователь выделяет новую дату, то обновляется эта информация в текстовом представлении, расположенном над выборщиком.

Вы можете скомпилировать это приложение, приведенное в листингах 10.19—10.23, с помощью SDK, создав проект приложения DatePicker на базе представления. Если вы хотите увидеть, как создавать все эти объекты с нуля, то удалите код Interface Builder.

Рис. 10.7. Пример DatePicker

Листинг 10.19. Прототипы делегата приложения DatePicker (DatePickerAppDelegate.h)

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

@class DatePickerViewController;

@interface DatePickerAppDelegate : NSObject <UIApplicationDelegate> {
    UIWindow *window;
    DatePickerViewController *viewController;
}

@property (nonatomic, retain) IBOutlet UIWindow *window;
@property (nonatomic, retain) IBOutlet DatePickerViewController
  *viewController;

@end
```

Листинг 10.20. Делегат приложения DatePicker (DatePickerAppDelegate.m)

```
#import "DatePickerAppDelegate.h"
#import "DatePickerViewController.h"
```

```
@implementation DatePickerAppDelegate

@synthesize window;
@synthesize viewController;

- (void)applicationDidFinishLaunching:(UIApplication *)application {
    CGRect screenBounds = [ [ UIScreen mainScreen ] bounds ];

    self.window = [ [ [ UIWindow alloc ] initWithFrame: screenBounds ]
      autorelease ];
    viewController = [ [ DatePickerViewController alloc ] init ];

    [ window addSubview: viewController.view ];
    [ window makeKeyAndVisible ];
}

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

@end
```

Листинг 10.21. Прототипы контроллера представления DatePicker (DatePickerViewController.h)

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

@protocol UIPickerViewDataSource, UIPickerViewDelegate;

@interface DatePickerViewController : UIViewController {
    UIDatePicker *pickerView;
    UITextView *textView;
    NSDate *minDate, *maxDate;
}
- (void) dateChanged: (id)sender;

@end
```

Листинг 10.22. Контроллер представлений DatePicker (DatePickerViewController.m)

```
#import "DatePickerViewController.h"

@implementation DatePickerViewController

- (id)init {
    self = [ super init ];
    if (self != nil) {
        minDate = [ [ NSDate alloc ]
            initWithString: @"1773-12-16 12:00:00 -0500" ];
        maxDate = [ [ NSDate alloc ]
            initWithString: @"1776-07-04 12:00:00 -0500" ];
    }
    return self;
}

- (void)loadView {
    CGRect bounds = [ [ UIScreen mainScreen ] applicationFrame ];

    [ super loadView ];

    pickerView = [ [ UIDatePicker alloc ] initWithFrame:
        CGRectMake(0.0, bounds.size.height - 216.0, 0.0, 0.0) ];

    pickerView.minimumDate = minDate;
    pickerView.maximumDate = maxDate;
    pickerView.datePickerMode = UIDatePickerModeDate;
    pickerView.date = minDate;

    [ pickerView addTarget: self action:
     @selector(dateChanged:)
          forControlEvents:UIControlEventValueChanged
    ];

    [ self.view addSubview: pickerView ];
```

```
    textView = [ [ UITextView alloc ] initWithFrame:
      CGRectMake(0.0, 0.0, bounds.size.width,
                           bounds.size.height - 216.0) ];
    textView.font = [ UIFont fontWithName: @"Arial" size: 22.0 ];
    textView.textColor = [ UIColor redColor ];
    textView.textAlignment = UITextAlignmentCenter;
    textView.text = [ pickerView.date description ];
    textView.editable = NO;

    [ self.view addSubview: textView ];
}

- (void) dateChanged: (id)sender
{
    UIDatePicker *control = (UIDatePicker *) sender;
    NSDate *selectedDate = control.date;
    textView.text = [ selectedDate description ];
}

- (BOOL)shouldAutorotateToInterfaceOrientation:
  (UIInterfaceOrientation)interfaceOrientation
{
    return (interfaceOrientation == UIInterfaceOrientationPortrait);
}

- (void)didReceiveMemoryWarning {
    [ super didReceiveMemoryWarning ];
}

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

@end
```

Листинг 10.23. Функция `main` для DatePicker (main.m)

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

int main(int argc, char *argv[]) {
    NSAutoreleasePool * pool = [[NSAutoreleasePool alloc] init];
    int retVal = UIApplicationMain(argc, argv, nil,
                                   @"DatePickerAppDelegate");
    [pool release];
    return retVal;
}
```

Как это работает

1. Когда приложение запущено, делегат приложения создает новый контроллер представлений, который, в свою очередь, задает объекты `NSDate`, содержащие минимальную и максимальную даты.
2. Когда вызывается метод `loadView` контроллера представлений, создаются объекты `UIDatePicker` и `UITextView`, которые добавляются к контроллеру представлений в качестве подчиненных представлений. Выборщик даты инициализируется с минимальной и максимальной датами, а текущее значение устанавливается как минимальное значение. Добавляется наблюдатель, чтобы вызывать метод `dateChanged` всякий раз, когда пользователь будет выбирать новую дату.
3. Внутренние структуры класса `UIDatePicker` создают выборщика, основываясь на заданных датах и стилях.
4. Когда пользователь выбирает на циферблате выборщика новый элемент, структура уведомлений `UIControl` вызывает целевой метод `dateChanged`. Он считывает новую дату и отображает ее в текстовом представлении, расположенном над выборщиком дат.

Для дальнейшего изучения

Когда вы покончите с вращением циферблата, проверьте наличие прототипа UIDatePicker.h. Вы найдете его в папке /Developer/Platforms/iPhoneOS.platform внутри каталога Headers библиотеки UI Kit.

Панели вкладок

Панели вкладок (tab bars) — это одно из решений Apple для универсальных устройств, не имеющих физических кнопок. Хотя приложения iPhone обладают достаточно богатой функциональностью, тем не менее, многие из них имеют по четыре-пять важных функций, доступ к которым пользователю необходимо получать незамедлительно. Расположенные вдоль нижней части окна панели вкладок предоставляют то, что на других мобильных устройствах обычно называется кнопками быстрого доступа. Если возвратиться к сравнению Apple с книгой, то можно сказать, что панели вкладок являются закладками (bookmarks) к различным главам приложения. Они также помогают разработчику логически группировать функции внутри приложения, чтобы удобно предоставлять основную функциональность приложения.

Многие из встроенных приложений iPhone, включая приложение Mobile Phone, YouTube и iTunes WiFi Music Store, используют панели вкладок. Вкладки разделяют связанные страницы данных (например, Featured Music, Purchased Songs и т. д.) или предоставляют быстрый доступ к различным функциям в рамках одного приложения (например, Contacts, Recent Calls, Voicemail).

Контроллеры панели вкладок

В UI Kit класс `UITabBar` представляет панели вкладок, а класс `UITabBarController` инкапсулирует их, что облегчает управление множеством различных представлений и переходов. Как и остальные контроллеры представлений, панели вкладок созданы быть автономными в своем представлении. Внутри же они обрабатывают весь поток выбора кнопок и переходов представлений — они "просто работают".

Контроллеры панелей вкладок работают схожим с контроллерами навигации образом: контроллеры навигации считывают свойства панели навигации каждого класса контроллера представлений, протолкнутого в стек. Контроллер представлений сам по себе описывает свойства своей панели, которые отображаются при отображении представления. Аналогично контроллер панели вкладок позволяет вам добавлять к нему целый ряд контроллеров представлений, и каждый контроллер представлений определяет собственные свойства, которые затем отображаются контроллером панели вкладок.

Рассматривайте такой подход как массовое производство коробок с цветными карандашами. Каждый карандаш, помещаемый вами в коробку, имеет

собственные цвет и надпись. Если вы напечатаете список существующих цветов на коробке, то вам придется менять коробку всякий раз, когда вы будете выпускать другой набор цветов. Более рентабельный способ спроектировать коробку — это поместить пустое пластиковое окошко на передней стороне коробки, чтобы покупатель мог видеть все цвета, имеющиеся внутри коробки. Это позволит вам менять набор цветов карандашей, не меняя при этом коробки.

Каждое представление, которое вы хотите отобразить, подобно одному цветному карандашу, а контроллер панели вкладок — коробке. Вместо того чтобы жестко кодировать в контроллере панели вкладок названия и изображения кнопок, контроллер панели вкладок просто подготавливает каждое представление так, чтобы его собственная "надпись" (кнопка панели вкладок) была видна сквозь коробку. Тем самым надписи принадлежат представлениям, а не контроллеру. Затем контроллер панели вкладок управляет всей работой по переходу к выбранному представлению, когда пользователь нажимает соответствующую ему кнопку.

Создание контроллера панели вкладок

Чтобы создать контроллер панели вкладок, сначала вам понадобятся отдельные страницы, для которых вы хотите предоставить кнопки. Создайте каждую страницу как объект `UIViewController`, как вы это делаете еще с *главы 3*.

Создание коллекции

Для обработки различных функций в рамках вашего приложения у вас может иметься несколько различных типов контроллеров представлений. Если вы пишете игровое приложение, то у вас могут быть такие контроллеры, как `GameViewController`, `SettingsViewController`, `HighScoreViewController` и `AboutViewController`. Прежде чем создать панель вкладок, создайте массив объектов контроллера представлений, которые вы хотите отобразить на панели вкладок. Вот пример этого:

```
NSMutableArray *viewControllers = [ [ NSMutableArray alloc ] init ];
[ viewControllers addObject: myGameViewController ];
[ viewControllers addObject: mySettingsViewController ];
[ viewControllers addObject: myHighScoreViewController ];
[ viewControllers addObject: myAboutViewController ];
```

Настройка свойств кнопок

Как вы узнали из примера с коробкой цветных карандашей, каждый контроллер представлений имеет собственную "надпись", определяющую то, как должна выглядеть кнопка панели вкладок. Затем панель вкладок отображает кнопку панели вкладок в соответствии со свойствами контроллера представлений. В рамках метода `init` контроллера представлений настраивается кнопка панели вкладок, чтобы описать ее название и/или свойство `tabBarItem`:

```
- (id) init {
    self = [ super init ];
    if (self != nil) {
        self.tabBarItem = [ [ UITabBarItem alloc ]
            initWithTitle: @"High Scores"
            image: [ UIImage imageNamed: @"high_scores.png"
            tag: 4
        ];
    }
    return self;
}
```

Установите свойство `tabBarItem` в объект `UITabBarItem`. Вы можете инициализировать элементы панели вкладок одним из двух способов. Метод `initWithTitle`, как показано в предыдущем примере, позволяет вам использовать название и изображение для формирования кнопки с задаваемыми пользователем данными. Кроме того, существует метод для создания целого ряда системных кнопок — стандартных кнопок, широко применяемых во многих приложениях iPhone. Чтобы создать такие системные кнопки, воспользуйтесь методом `initWithTabBarSystemItem`:

```
self.tabBarItem = [ [ UITabBarItem alloc ]
    initWithTabBarSystemItem: UITabBarSystemItemFavorites tag: 4
];
```

Вы можете использовать следующие идентификаторы системных кнопок:

- `UITabBarSystemItemMore`;
- `UITabBarSystemItemFavorites`;
- `UITabBarSystemItemFeatured`;
- `UITabBarSystemItemTopRated`;
- `UITabBarSystemItemRecents`;

- `UITabBarSystemItemContacts`;
- `UITabBarSystemItemHistory`;
- `UITabBarSystemItemBookmarks`;
- `UITabBarSystemItemSearch`;
- `UITabBarSystemItemDownloads`;
- `UITabBarSystemItemMostRecent`;
- `UITabBarSystemItemMostViewed`.

Создание контроллера панели вкладок

После того как каждому представлению будет сопоставлена кнопка панели вкладок, вы можете создать объект контроллера панели вкладок и добавить к нему массив контроллеров представлений:

```
UITabBarController *tabBarController = [[UITabBarController alloc] init];
tabBarController.viewControllers = viewControllers;
```

Отображение контроллера панели вкладок

После создания контроллера панели вкладок отобразите его, добавив его к окну или другому представлению как подчиненное представление:

```
[ window addSubview: tabBarController.view ];
[ window makeKeyAndVisible ];
```

Настраиваемые кнопки

По умолчанию контроллер панели вкладок позволяет пользователю настраивать расположение кнопок, если этих кнопок больше пяти. Это можно сделать, щелкнув по вкладке **More**, расположенной на панели навигации за кнопкой **Edit**. Вы можете разрешить настройку только определенных вкладок, или же вообще запретить любую настройку. Для этого установите `customizableViewControllers` контроллера панели вкладок в массив контроллеров представлений, который вы хотите разрешить настраивать пользователю:

```
NSMutableArray *customizableControllers = [[NSMutableArray alloc] init];
[ customizableControllers addObject: myGameViewController ];
```

```
[ customizableControllers addObject: mySettingsViewController ];
tabBarController.customizableViewControllers = customizableControllers;
```

Для запрета любой настройки используйте `nil`:

```
tabBarController.customizableViewControllers = nil;
```

Навигация

Когда вы отображаете контроллер панели вкладок, то этот контроллер управляет собственной навигацией, так что выбранное представление вкладки автоматически переносится на передний план экрана. Если к контроллеру панели вкладок прикреплено более пяти контроллеров представлений, то контроллер панели вкладок автоматически отобразит вкладку с меткой **More**. Когда пользователь коснется этой кнопки, контроллеры представлений, которые не смогут поместиться на экране, будут отображены в виде табличного списка. Чтобы считать или изменить активный в данный момент контроллер представлений, воспользуйтесь свойством `selectedViewController`:

```
tabBarController.selectedViewController = myGameViewController;
UIViewController *activeController =
  tabBarController.selectedViewController;
if (activeController == myGameViewController) {
    /* В текущий момент активен myGameViewController */
}
```

Также вы можете ссылаться на кнопки по их порядковому номеру:

```
tabBarController.selectedIndex = 3;
```

> ***Примечание***
>
> Если вы задали более пяти контроллеров представлений, то контроллеру представлений "More" будет присвоен собственный порядковый номер.

Делегированные действия

Чтобы получать уведомления, когда на панели вкладок выбрано то или иное представление, назначьте контроллеру панели вкладок делегата:

```
tabBarController.delegate = self;
```

Данный делегат будет уведомляться всякий раз, когда выбирается контроллер представлений, посредством вызова метода-делегата `didSelectViewController`:

```
- (void)tabBarController:(UITabBarController *)tabBarController
    didSelectViewController:(UIViewController *)viewController
{
    /* Специальный код для обработки выбора */
}
```

Этот делегат также будет уведомляться всякий раз, когда пользователь будет завершать настройку раскладки панели вкладок. Вы сможете получать эти уведомления посредством добавления метода-делегата `didEndCustomizingViewControllers`:

```
- (void)tabBarController:(UITabBarController *)tabBarController
    didEndCustomizingViewControllers:(NSArray *)viewControllers
    changed:(BOOL)changed
{
    /* Специальный код для обработки завершения настройки панели вкладок
*/
}
```

Еще один способ реализации текстовой книги: TabDemo

В *главе 3* вы познакомились с переходами, реализованными в примере с переворачиванием страниц. Данный пример схож с тем примером, за исключением того, что вы будете использовать восемь страниц, соответствующих восьми различным контроллерам представлений в приложении. Каждая страница будет управляться кнопкой на панели вкладок, которая при нажатии будет перепрыгивать к соответствующей странице. Панель вкладок автоматически обработает переход к выбранному контроллеру представлений.

Вы можете скомпилировать это приложение, приведенное в листингах 10.24—10.28, с помощью SDK, создав проект приложения TabDemo на базе представления. Если вы хотите увидеть, как создавать все эти объекты с нуля, то удалите код Interface Builder.

Листинг 10.24. Прототипы делегата приложения TabDemo (TabDemoAppDelegate.h)

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

@class TabDemoViewController;

@interface TabDemoAppDelegate : NSObject <UIApplicationDelegate> {
    UIWindow *window;
    UITabBarController *tabBarController;
    NSMutableArray *viewControllers;
}

@property (nonatomic, retain) IBOutlet UIWindow *window;

@end
```

Листинг 10.25. . Делегат приложения TabDemo (TabDemoAppDelegate.m)

```
#import "TabDemoAppDelegate.h"
#import "TabDemoViewController.h"

@implementation TabDemoAppDelegate

@synthesize window;

- (void)applicationDidFinishLaunching:(UIApplication *)application {
    CGRect screenBounds = [ [ UIScreen mainScreen ] bounds ];

    self.window = [ [ [ UIWindow alloc ] initWithFrame: screenBounds ]
        autorelease
    ];

    viewControllers = [ [ NSMutableArray alloc ] init ];
    for(int i = 0; i < 8; i ++) {
        [ viewControllers addObject: [
```

```
            [ TabDemoViewController alloc ] initWithPageNumber: i ]
    ];
}

    tabBarController = [ [ UITabBarController alloc ] init ];
    tabBarController.viewControllers = viewControllers;

    [ window addSubview: tabBarController.view ];
    [ window makeKeyAndVisible ];
}

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

@end
```

Листинг 10.26. Прототипы контроллера представления TabDemo (TabDemoViewController.h)

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

@interface TabDemoViewController : UIViewController {
    UITextView *textView;
    int page;
}
- (id) initWithPageNumber:(int)pageNumber;

@end
```

Листинг 10.27. Контроллер представлений TabDemo (TabDemoViewController.m)

```
#import "TabDemoViewController.h"

@implementation TabDemoViewController
```

```
- (id) initWithPageNumber:(int)pageNumber {
    self = [ super init ];
    if (self != nil) {
        page = pageNumber;
        self.title = [ NSString stringWithFormat: @"Page %d", page ];
        self.tabBarItem = [ [ UITabBarItem alloc ] initWithTitle:
            [ NSString stringWithFormat: @"Page %d", page ] image:
              nil tag: page ];
    }
    return self;
}

- (void) loadView {
    [ super loadView ];

    CGRect bounds = [ [ UIScreen mainScreen ] applicationFrame ];
    textView = [ [ UITextView alloc ] initWithFrame: bounds ];
    textView.text = [ [ NSString alloc ]
        initWithFormat: @"Text for page %d", page
    ];

    self.view = textView;
}

- (BOOL)shouldAutorotateToInterfaceOrientation:
    (UIInterfaceOrientation)interfaceOrientation
{
    return (interfaceOrientation == UIInterfaceOrientationPortrait);
}

- (void)didReceiveMemoryWarning {
    [ super didReceiveMemoryWarning ];
}

- (void)dealloc {
    [ textView release ];
```

```
    [ super dealloc ];
}

@end
```

Листинг 10.28. Функция `main` для TabDemo (main.m)

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

int main(int argc, char *argv[]) {

    NSAutoreleasePool * pool = [[NSAutoreleasePool alloc] init];
    int retVal = UIApplicationMain(argc, argv, nil,
                                   @"TabDemoAppDelegate");
    [pool release];
    return retVal;
}
```

Как это работает

1. Когда приложение запущено, его делегат создает набор из восьми различных контроллеров представлений и добавляет их в массив `viewControllers`. Класс `TabDemoViewController` использует собственный метод `initWithPageNumber`, который настраивает его название, кнопку и отображает текст в соответствии с передаваемым в него номером страницы.
2. Создается `UITabBarController` и присваивается массиву из восьми компонентов представления. Затем к основному окну добавляется контроллер панели вкладок.
3. Всякий раз, когда нажимается новая вкладка, контроллер панели вкладок автоматически обрабатывает переход к новому представлению. Также он отображает кнопку **More** и позволяет пользователю настраивать положения кнопок. Вот и все! Все очень просто.

Для дальнейшего изучения

Теперь, когда вы узнали о том, как заставить людей нажимать на ваши кнопки, поиграйте немного с панелями вкладок.

- Попробуйте создать системные кнопки и собственные изображения кнопок.

- Измените данный пример так, чтобы при выборе пользователем страницы 3 автоматически снималось выделение со страницы 4, просто чтобы запутать его.
- Проверьте наличие прототипов UITabBar.h, UITabBarController.h и UITabBarItem.h. Вы найдете их в папке /Developer/Platforms/iPhoneOS.platform внутри каталога Headers библиотеки UI Kit.

Показания сенсоров и информация об устройстве

iPhone обладает множеством сенсоров, таких как акселерометр (accelerometer) и датчик приближения (proximity sensor). Остальные сенсоры, как, например, датчик ориентации, используются для автоматического управления поворотами экрана, однако могут пригодиться и в заказных приложениях. Помимо сенсоров, класс `UIDevice` позволяет считывать определенную информацию уровня операционной системы, например, модель устройства, версию программного обеспечения и уникальный идентификатор.

Считывание ориентации

Хотя большинство классов контроллеров представлений выполняет всю работу по изменению пользовательского интерфейса устройства при перемене ориентации устройства, заказные приложения могут сами обрабатывать изменение ориентации. Сенсор ориентации может быть считан посредством класса `UIDevice`, который предоставляет набор простых перечислимых значений для идентификации ориентации. Чтобы считать ориентацию, получите экземпляр текущего объекта `UIDevice`:

```
UIDevice *device = [ UIDevice currentDevice ];
```

Получение доступа к ориентации из свойства ориентации объекта:

```
UIDeviceOrientation = device.orientation;
```

В качестве ориентации можно установить следующие значения:

- `UIDeviceOrientationUnknown` — перехват всех ошибок или аппаратных сбоев;
- `UIDeviceOrientationPortrait` — устройство ориентировано прямо по вертикали в книжном режиме;

- `UIDeviceOrientationPortraitUpsideDown` — устройство ориентировано вверх дном по вертикали в книжном режиме;
- `UIDeviceOrientationLandscapeLeft` — устройство повернуто против часовой стрелки в альбомном режиме;
- `UIDeviceOrientationLandscapeRight` — устройство повернуто по часовой стрелке в альбомном режиме;
- `UIDeviceOrientationFaceUp` — устройство лежит на ровной поверхности, например на столе, лицевой стороной вверх;
- `UIDeviceOrientationFaceDown` — устройство лежит на ровной поверхности, например на столе, лицевой стороной вниз.

Считывание информации об устройстве

Помимо ориентации устройства вы можете считать еще и другую информацию из экземпляра `UIDevice`. Для этого сначала получите экземпляр текущего устройства:

```
UIDevice *device = [ UIDevice currentDevice ];
```

Потом вы сможете считать приведенные в табл. 10.11 свойства. Каждое свойство для описания устройства предоставляет объект `NSString`.

Таблица 10.11

Свойство	Описание
`name`	Имя, присвоенное iPhone его владельцем
`model`	Модель устройства; "iPhone", "iPod Touch" и т. д.
`localizationModel`	Локализованная версия модели
`systemName`	Название ОС; "iPhone OS"
`systemVersion`	Версия операционной системы
`uniqueIdentifier`	Уникальный идентификатор устройства

Считывание показаний акселерометра

API ориентации получает всю информацию от небольшого акселерометра, встроенного в iPhone. Эта крошечная часть аппаратного обеспечения сооб-

щает координаты *X*, *Y*, *Z* положения устройства. В то время как считывание ориентации может дать вам общее представление о том, в каком положении держится устройство, то считывание акселерометра позволяет вам уловить самые незаметные перемещения этого устройства. Очень часто это используется в играх, когда пользователь может перемещать устройство, чтобы управлять каким-либо игровым персонажем или трясти устройство при выбрасывании костей или очистки панели рисования.

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, таких как незамысловатые игры, пупсы (bobble heads) и в программах рисования.

> ***Примечание***
>
> Помните, что не все ваши заказчики заметят и оценят акселерометр iPhone. Добавление возможности, разрешающей использование элементов управления с множественными касаниями, поможет вам увеличить аудиторию ваших потенциальных покупателей.

Чтобы считать показания акселерометра, получите общий экземпляр объекта `UIAccelerometer`, который доступен для вашего приложения:

```
UIAccelerometer *accelerometer = [ UIAccelerometer sharedInstance ];
```

Теперь вы можете получить доступ к координатам *X*, *Y*, *Z* акселерометра посредством его соответствующих свойств:

```
UIAccelerationValue x = accelerometer.x;
UIAccelerationValue y = accelerometer.y;
UIAccelerationValue z = accelerometer.z;
```

Тип данных `UIAccelerationValue` — с плавающей точкой двойной точности (double floating point).

Отслеживание перемещений

Чтобы отслеживать перемещения акселерометра, ваше приложение должно получать уведомления всякий раз, когда ваше устройство меняет скорость. Для этого назначьте какой-либо объект в качестве делегата акселерометра:

```
accelerometer.delegate = self;
```

Теперь ваш класс-делегат будет уведомляться всякий раз, когда устройство будет менять скорость:

```
- (void)accelerometer:(UIAccelerometer *)accelerometer
didAccelerate:(UIAcceleration *)acceleration {
    /* Код для обработки акселерации */
}
```

Наконец, вы можете установить свойство `updateInterval` с `NSTimeInterval`, чтобы задать временные интервалы, через которые вы хотите получать обновления от акселерометра:

```
NSTimeInterval timeInterval = 0.5;
accelerometer.updateInterval = timeInterval;
```

> ***Внимание!***
>
> Не задавайте слишком низкие интервалы обновления, иначе ваше приложение потонет в вызовах делегата. Apple задала минимальное встроенное значение, но этот минимум не опубликован.

Датчик приближения

Если приложение выполняет какую-либо задачу, которая обычно требует, чтобы пользователь держал устройство у своего лица, то должен быть активирован датчик приближения, чтобы iPhone при этом выключал свой экран.

Чтобы активировать датчик приближения, получите общий экземпляр вашего объекта `UIApplication` и доступ к его свойству `proximitySensingEnabled`:

```
UIApplication *myApp = [ UIApplication sharedApplication ];
myApp.proximitySensingEnabled = YES;
```

В активном состоянии датчик приближения будет автоматически выключать экран, когда он почувствует приближение лица. На сегодня не существует санкционированного метода-делегата или других удобных средств SDK для уведомления приложения о таких ситуациях. Apple должна озадачить своих разработчиков, чтобы они с этим что-либо сделали.

> ***Примечание***
>
> Убедитесь в том, что датчик приближения отключен, если устройство больше не должно находиться у лица пользователя.

Для дальнейшего изучения

Проверьте наличие прототипов UIDevice.h и UIAccelerometer.h. Вы найдете их в папке /Developer/Platforms/iPhoneOS.platform внутри каталога Headers библиотеки UI Kit.

Представления прокрутки

Рассматривайте представления прокрутки (scroll views) как дешифрующий элемент, который можно найти в пачках с крупами. Поместив этот маленький элемент из пластика на какую-либо часть секретной шифровальной 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 содержимого скрыта из поля зрения за пределами экрана до тех пор, пока пользователь не переместит окно к той части, которую он хочет увидеть. Представления прокрутки позволяют вам не только прокручивать содержимое, но и еще увеличивать или уменьшать его, и даже переворачивать страницы.

Создание представления прокрутки

Класс `UIScrollView` ответственен за всю прокрутку в UI Kit. Создание представления прокрутки эквивалентно созданию дешифрующего элемента и пустых страниц в секретной шифровальной книге:

```
CGRect bounds = [ [ UIScreen mainScreen ] applicationFrame ];
UIScrollView *scrollView = [[UIScrollView alloc] initWithFrame: bounds];
```

После создания представления прокрутки вы склеиваете содержимое другого представления с пустыми страницами представления прокрутки. Тем самым вы создаете окно прокручиваемого содержания:

```
[ scrollView addSubview: myBiggerView ];
```

Представлению прокрутки вы должны предоставить действительный размер содержимого так, чтобы оно знало, какой объем прокручивать:

```
scrollView.contentSize = myBiggerView.frame.size;
```

Чтобы разрешить масштабирование, настройте свойства `maximumZoomScale` и `minimumZoomScale`. Это позволит пользователю делать пинч (pinch) и изменять размер содержимого:

```
scrollView.maximumZoomScale = 3.00; /* Позволяет увеличивать масштаб
                                    /* в три раза от исходного размера */
scrollView.minimumZoomScale = 0.25; /* Позволяет уменьшать масштаб */
                                    /* на 25% от исходного размера */
```

Чтобы разрешить масштабирование, вам также потребуется добавить делегата `UIScrollViewDelegate`, который будет отвечать методу `viewForZoomingInScrollView`. Этот метод возвращает объект `UIView`, используемый при масштабировании содержимого:

```
scrollView.delegate = self;
- (UIView *)viewForZoomingInScrollView:(UIScrollView *)scrollView {
    return myBiggerView;
}
```

Примечание

Для данных большого размера разумно начать со шкалы масштабирования, меньшей действительного размера (1, 0), чтобы позволить пользователю плавно увеличивать масштаб.

Свойства

Помимо только что рассмотренных основных свойств, представления прокрутки обладают множеством других свойств, которые могут осуществлять тонкую настройку поведения вашего содержимого. Вы можете настраивать класс `UIScrollView` несколькими способами. Наиболее распространенными являются следующие свойства:

- `indicatorStyle` задает тип индикаторов полосы прокрутки, которую вы хотите использовать. По умолчанию отображаются черные полосы прокрутки с белой границей, которая подходит для большинства фонов. Существуют следующие стили:
  - `UIScrollViewIndicatorStyleDefault`;
  - `UIScrollViewIndicatorStyleBlack`;
  - `UIScrollViewIndicatorStyleWhite`;

- `contentOffset` — структура `CGPoint`, содержащая смещение содержимого для отображения в левом верхнем углу окна. По умолчанию начало лежит в точке (0, 0), но вы можете расположить ваше содержимое по-другому;
- `directionalLockEnabled`. По умолчанию пользователю разрешается осуществлять прокрутку одновременно по горизонтали и по вертикали. Установите это свойство в `YES`, чтобы запретить пользователю либо горизонтальную, либо вертикальную прокрутку в зависимости от исходного жеста;
- `bounces`. Когда пользователь достигнет края прокручиваемой области, то данная функциональность позволит пользователю перетащить область слегка за пределы границ. Когда пользователь отпустит свой палец, эта область отскочит на свое место как резиновая лента, давая пользователю визуальное понимание того, что он достиг начала или конца документа. Если вы не хотите, чтобы пользователь мог прокручивать назад просматриваемое содержимое, установите это свойство в `NO`;
- `bouncesZoom`. Как и вариант `bounce`, этот метод позволяет пользователю масштабировать за пределы минимального и максимального уровней масштабирования, а затем вовзращать пользователя обратно в пределы области. Если вы не хотите, чтобы пользователь мог масштабировать прошлые области, заданные вами, установите это свойство в `NO`;
- `pagingEnabled`. При разрешенном пейджинге (paging) представление прокрутки разбивается на отдельные сегменты, а у пользователя появляется ощущение перелистывания страниц. Вы можете использовать это для выполнения *фликинга* (flicking) страниц, рассматриваемого в *главе 13*.

Методы-делегаты

Когда представлению прокрутки назначен делегат, во время специальных событий уведомляются следующие методы-делегаты:

- `- (void)scrollViewDidScroll:(UIScrollView *)scrollView;`

  Уведомляется, когда прокручивается представление. Включает указатель на то представление прокрутки, которое прокручивается, в этой точке может быть считано свойство `contentOffset`, чтобы определить, куда оно было прокручено.

- `- (void)scrollViewWillBeginDragging:(UIScrollView *)scrollView;`

  Уведомляется, когда пользователь впервые приступает к перетаскиванию представления в любом направлении. Этот метод также может быть использован для считывания свойства `contentOffset` представления прокрутки, чей указатель передается ему в качестве аргумента.

- `- (void)scrollViewDidEndDragging:(UIScrollView *)scrollView willDecelerate: (BOOL)decelerate;`

  Уведомляется, когда пользователь отпускает перетаскивающий палец. Также предоставляется булево значение, указывающее на то, нужно ли замедлять представление прокрутки до сообщения итогового положения.

- `- (void)scrollViewWillBeginDecelerating:(UIScrollView *)scrollView;`

  Уведомляется, когда пользователь поднимает свой палец, в то время как представление прокрутки продолжает перемещаться. Это может применяться для считывания свойства `contentOffset`, чтобы специально определить то, куда в последний раз прокрутил пользователь, прежде чем поднять свой палец, даже если это будет не последнее положение панелей прокрутки.

- `- (void)scrollViewDidEndDecelerating:(UIScrollView *)scrollView;`

  Уведомляется, когда завершается упомянутое выше замедление, а представление прокрутки останавливает прокрутку. К моменту получения уведомления свойство `contentOffset` будет отражать итоговое положение панели прокрутки.

- `- (void)scrollViewDidEndZooming:(UIScrollView *)scrollView withView:(UIView *) view atScale:(float)scale;`

  Уведомляется, когда пользователь уменьшил/увеличил содержимое до заданного масштаба. В качестве аргумента передается шкала, представленная значением с плавающей точкой двойной точности.

- `- (BOOL)scrollViewShouldScrollToTop:(UIScrollView *)scrollView;`
  `(void)scrollViewDidScrollToTop:(UIScrollView *)scrollView;`

  Когда пользователь касается строки состояния iPhone, делегат представления прокрутки может определить, должно ли представление прокрутиться обратно наверх.

Прокрутка метеорологической карты: BigImage

Этот пример использует объект `NSData`, о котором вы уже знаете, для загрузки большой метеорологической карты и отображения пользователю ее представления прокрутки. Вы увидите, как другой класс `UIView` (`UIImageView`) прикрепляется к представлению прокрутки и как работает его делегат.

Вы можете скомпилировать это приложение, представленное в листингах 10.29—10.33, с помощью SDK, создав проект оконного приложения BigImage. Если вы хотите увидеть, как создавать все эти объекты с нуля, то удалите код Interface Builder.

Листинг 10.29. Прототипы делегата приложения BigImage (BigImageAppDelegate.h)

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

@class BigImageViewController;

@interface BigImageAppDelegate : NSObject <UIApplicationDelegate> {
    UIWindow *window;
    BigImageViewController *viewController;
}

@property (nonatomic, retain) IBOutlet UIWindow *window;
@property (nonatomic, retain) IBOutlet BigImageViewController
  *viewController;

@end
```

Листинг 10.30. Делегат приложения BigImage (BigImageAppDelegate.m)

```
#import "BigImageAppDelegate.h"
#import "BigImageViewController.h"

@implementation BigImageAppDelegate

@synthesize window;
@synthesize viewController;
```

```
- (void)applicationDidFinishLaunching:(UIApplication *)application {
    CGRect screenBounds = [ [ UIScreen mainScreen ] bounds ];

    self.window = [ [ [ UIWindow alloc ] initWithFrame: screenBounds ]
      autorelease ];
    viewController = [ [ BigImageViewController alloc ] init ];

    [ window addSubview: viewController.view ];
    [ window makeKeyAndVisible ];
}

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

@end
```

Листинг 10.31. Прототипы контроллера представлений BigImage (BigImageViewController.h)

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

@interface BigImageViewController : UIViewController
  <UIScrollViewDelegate> {
    UIScrollView *scrollView;
    UIImageView *imageView;
}

@end
```

Листинг 10.32. Контроллер представлений BigImage (BigImageViewController.m)

```
#import "BigImageViewController.h"

@implementation BigImageViewController
```

```
- (id)init {
    self = [ super init ];
    if (self != nil) {
        imageView = [ [ UIImageView alloc ] initWithImage:
            [ UIImage imageWithData:
            [ NSData dataWithContentsOfURL:
               [ NSURL URLWithString:
                   @"http://forecast.weather.gov/wwamap/png/US.png" ]
            ]
        ] ];
    }
    return self;
}

- (void)loadView {
    [ super loadView ];
    CGRect bounds = [ [ UIScreen mainScreen ] applicationFrame ];

    scrollView = [ [ UIScrollView alloc ] initWithFrame: bounds ];
    scrollView.contentSize = imageView.frame.size;
    scrollView.maximumZoomScale = 3.0;
    scrollView.minimumZoomScale = 0.25;
    scrollView.delegate = self;
    scrollView.bounces = NO;
    [ scrollView addSubview: imageView ];

    self.view = scrollView;
}

- (UIView *)viewForZoomingInScrollView:(UIScrollView *)scrollView {
    return imageView;
}

- (BOOL)shouldAutorotateToInterfaceOrientation:
    (UIInterfaceOrientation)interfaceOrientation
```

```
{
    return (interfaceOrientation == UIInterfaceOrientationPortrait);
}

- (void)didReceiveMemoryWarning {
    [ super didReceiveMemoryWarning ];
}

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

@end
```

Листинг 10.33. Функция `main` для BigImage (main.m)

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

int main(int argc, char *argv[]) {
    NSAutoreleasePool * pool = [[NSAutoreleasePool alloc] init];
    int retVal = UIApplicationMain(argc, argv, nil,
                                   @"BigImageAppDelegate");
    [pool release];
    return retVal;
}
```

Как это работает

1. Когда приложение запущено, его делегат, как обычно, создает контроллер представлений и прикрепляет его к окну. Контроллер представлений инициализируется, что приводит к созданию объекта `UIImageView`. Содержимое представления изображения загружается с национального метеорологического сервиса.
2. Когда контроллер представлений загрузится, создастся объект `UIScrollView`. Представление изображения прикрепляется к представле-

нию прокрутки, и задаются свойства масштабирования и внешнего вида представления прокрутки.

3. Внутренние структуры представления прокрутки управляют всем процессом прокрутки, отображением панели прокрутки и другими связанными функциями.
4. Когда пользователь осуществляет масштабирование, вызывается метод-делегат представления прокрутки `viewForZoomingInScrollView`. Он возвращает тот же самый объект `UIImageView`, поскольку у нас нет копии карты более высокого разрешения.

Для дальнейшего изучения

Представления прокрутки являются легким способом уменьшать масштаб данных большого объема, представляемых пользователю.

- ❒ Попробуйте создать представления прокрутки для какого-либо вашего собственного содержимого.
- ❒ Проверьте наличие прототипа UIScrollView.h. Вы найдете его в папке /Developer/Platforms/iPhoneOS.platform внутри каталога Headers библиотеки UI Kit.

Web-представления

В *главе 3* мы познакомились с объектом `UITextView` и его скрытым методом `setHTML` для создания окон с HTML-форматированием. Вы можете использовать класс `UIWebView` для создания страницы в силе обозревателя, предоставляя многие из основных процедур, которые имеются в обозревателе Интернета: удаленная выборка страниц, прямая и обратная навигация, а также изменение масштаба. Он также полезен при отображении обогащенного текста (rich text) с использованием различных гарнитур и размеров шрифтов. Web-представление (web view) может даже распознавать телефонные номера на Web-страницах, разрешая пользователю дотрагиваться до них, чтобы выполнить телефонный звонок. Web-представления являются одними из ключевых компонентов, которые заставляют работать Safari. Web-представления могут обрабатывать не только Web-страницы, они также могут отображать графические изображения и PDF-файлы, расположенные как локально, так и удаленно, исходя их предоставляемого им содержимого.

Создание Web-представления

Web-представление создается, как и любой другой класс `UIVIew`, с помощью метода `initWithFrame`:

```
CGRect bounds = [ [ UIScreen mainScreen ] applicationFrame ];
UIWebView *webView = [ [ UIWebView alloc ] initWithFrame: bounds ];
```

После создания Web-представления вы можете задать несколько свойств. Для автоматического масштабирования страницы, чтобы она помещалась на экране, установите свойство `scalesPageToFit` в `YES`:

```
webView.scalesPageToFit = YES;
```

Если вы хотите, чтобы Web-представление автоматически распознавало телефонные номера и позволяло пользователю дотрагиваться до них для совершения звонка, установите свойство `detectsPhoneNumbers` в `YES`:

```
webView.detectsPhoneNumbers = YES;
```

Отображение Web-представления

Чтобы отобразить Web-представление, прикрепите его к существующему окну или классу представления:

```
[ self.view addSubview: webView ];
```

Загрузка содержимого

После создания Web-представления вы можете загрузить содержимое множеством способов. Наиболее распространенным способом является загрузка содержимого с помощью метода `loadRequest` для загрузки локально или удаленно расположенного ресурса. Метод `loadRequest` принимает объект `NSURLRequest`, который может быть создан из существующего объекта `NSURL`. Вот пример создания и загрузки удаленной Web-страницы:

```
NSURL *url = [ NSURL URLWithString: @"http://www.oreilly.com" ];
NSURLRequest *request = [ NSURLRequest requestWithURL: url ];
[ webView loadRequest: request ];
```

Для загрузки локально расположенного файлового ресурса воспользуйтесь инициализатором `fileURLWithPath` класса `NSURL`:

```
NSURL *url = [ NSURL fileURLWithPath: filePath ];
NSURLRequest *request = [ NSURLRequest requestWithURL: url ];
[ webView loadRequest: request ];
```

Класс `UIWebView` также поддерживает загрузку в качестве источника объекта `NSString`. Вы можете при желании предоставить основной URL, чтобы указать объекту `UIWebView`, как переходить по ссылкам и загружать удаленные ресурсы:

```
 [ webView loadHTMLString: myHTML
    baseURL: [ NSURL URLWithString: @"http://www.mywebsite.com" ]
];
```

Навигация

Класс `UIWebView` управляет навигацией обозревателя. Вы можете контролировать переходы вперед и назад с помощью методов `goForward` и `goBack`:

```
[ webView goBack ];
[ webView goForward ];
```

Чтобы перезагрузить имеющееся содержимое, воспользуйтесь методом `reload` представления прокрутки:

```
[ webView reload ];
```

Чтобы отменить загрузку содержимого, воспользуйтесь методом `stopLoading`:

```
[ webView stopLoading ];
```

Методы-делегаты

Класс Web-представления поддерживает множество методов-делегатов `UIWebViewDelegate`. Эти методы уведомляются, когда происходят определенные события. Чтобы воспользоваться ими, назначьте делегата вашему Web-представлению:

```
webView.delegate = self;
```

Приводимые далее методы уведомляются Web-представлением. Каждый метод-делегат в качестве своего первого параметра предоставляет указатель на

Web-представление, что позволяет вам обслуживать несколько Web-представлений в рамках одного делегата.

- `- (BOOL)webView:(UIWebView *)webView,`
  `shouldStartLoadWithRequest:(NSURLRequest *)request,`
  `navigationType:(UIWebViewNavigationType)navigationType;`

  Уведомляется, когда Web-представлению указывается загрузить содержимое. Чтобы начать загрузку, он должен вернуть `YES`. Предоставляемый параметр навигационного типа ссылается на источник запроса и может быть одним из следующих:

  - `UIWebViewNavigationTypeLinkClicked`;
  - `UIWebViewNavigationTypeFormSubmitted`;
  - `UIWebViewNavigationTypeBackForward`;
  - `UIWebViewNavigationTypeReload`;
  - `UIWebViewNavigationTypeFormResubmitted`;
  - `UIWebViewNavigationTypeOther`.

- `- (void)webViewDidStartLoad:(UIWebView *)webView;`

  Уведомляется, когда Web-представление начинает загрузку запроса.

- `- (void)webViewDidFinishLoad:(UIWebView *)webView;`

  Уведомляется, когда Web-представление завершает загрузку запроса.

- `- (void)webView:(UIWebView *)webView,`
  `didFailLoadWithError:(NSError *)error;`

  Уведомляется, если в процессе загрузки запроса происходит ошибка. Для определения типа произошедшей ошибки предоставляется объект `NSError`.

Поисковая программа Google: WebDemo

Этот пример использует `UIWebView` для отображения результатов поиска Google. Введите поисковые термины в `UIView`-класс `UISearchBar`, который является простейшим классом для приема поискового запроса. К сожалению, Apple приватизировала `UISearchField` панели поиска, который Safari использует для отображения как адресного поля, так и поля поискового запроса в рамках одной панели навигации. В данном примере пользователь вводит

критерии поиска и нажимает кнопку **Search**. В результате этого осуществляет запрос к Google, использующий предоставленные данные, а результаты запроса отображаются в Web-представлении (рис. 10.8).

Рис. 10.8. Пример WebDemo

Вы можете скомпилировать это приложение, приведенное в листингах 10.34—10.38, с помощью SDK, создав проект приложения WebDemo на базе представления. Если вы хотите увидеть, как создавать все эти объекты с нуля, то удалите код Interface Builder.

Листинг 10.34. Прототипы делегата приложения WebDemo (WebDemoAppDelegate.h)

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


@class WebDemoViewController;
```

```
@interface WebDemoAppDelegate : NSObject <UIApplicationDelegate> {
    UIWindow *window;
    WebDemoViewController *viewController;
}

@property (nonatomic, retain) IBOutlet UIWindow *window;
@property (nonatomic, retain) IBOutlet WebDemoViewController
  *viewController;

@end
```

Листинг 10.35. Делегат приложения WebDemo (WebDemoAppDelegate.m)

```
#import "WebDemoAppDelegate.h"
#import "WebDemoViewController.h"

@implementation WebDemoAppDelegate

@synthesize window;
@synthesize viewController;

- (void)applicationDidFinishLaunching:(UIApplication *)application {
    CGRect screenBounds = [ [ UIScreen mainScreen ] bounds ];

    self.window = [ [ [ UIWindow alloc ] initWithFrame: screenBounds ]
      autorelease ];
    viewController = [ [ WebDemoViewController alloc ] init ];

    [ window addSubview: viewController.view ];
    [ window makeKeyAndVisible ];
}

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

@end
```

Листинг 10.36. Прототипы контроллера представлений WebDemo (WebDemoViewController.h)

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


@interface WebDemoViewController : UIViewController <UISearchBarDelegate>
{
    UISearchBar *searchBar;
    UIWebView *webView;
}


@end
```

Листинг 10.37. Контроллер представлений WebDemo (WebDemoViewController.m)

```
#import "WebDemoViewController.h"


@implementation WebDemoViewController


- (id)init {
    self = [ super init ];
    if (self != nil) {

    }
    return self;
}


- (void)loadView {
    [ super loadView ];
    CGRect bounds = [ [ UIScreen mainScreen ] applicationFrame ];

    searchBar = [ [ UISearchBar alloc ] initWithFrame:
        CGRectMake(0.0, 0.0, bounds.size.width, 48.0) ];
    searchBar.delegate = self;
    searchBar.placeholder = @"Google";
    [ self.view addSubview: searchBar ];
```

```
    webView = [ [ UIWebView alloc ] initWithFrame:
      CGRectMake(0.0, 48.0, bounds.size.width,
                            bounds.size.height - 48.0) ];
    webView.scalesPageToFit = YES;
    [ self.view addSubview: webView ];
}

- (void)searchBarSearchButtonClicked:(UISearchBar *)activeSearchBar {
    NSString *query = [ searchBar.text
        stringByReplacingOccurrencesOfString: @" " withString: @"+" ];
    NSURL *url = [ NSURL URLWithString:
        [ NSString stringWithFormat:
           @"http://www.google.com/search?q=%@", query ] ];
    NSURLRequest *request = [ NSURLRequest requestWithURL: url ];
    [ webView loadRequest: request ];
}

- (BOOL)shouldAutorotateToInterfaceOrientation:(UIInterfaceOrientation)
interfaceOrientation {
    return (interfaceOrientation == UIInterfaceOrientationPortrait);
}

- (void)didReceiveMemoryWarning {
    [ super didReceiveMemoryWarning ];
}

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

@end
```

Листинг 10.38. Функция `main` для WebDemo (main.m)

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

int main(int argc, char *argv[]) {

    NSAutoreleasePool * pool = [[NSAutoreleasePool alloc] init];
```

```
    int retVal = UIApplicationMain(argc, argv, nil,
                                   @"WebDemoAppDelegate");
    [pool release];
    return retVal;
}
```

Как это работает

1. Когда приложение запущено, его делегат, как обычно, создает контроллер представлений и прикрепляет его к окну. Контроллер представлений инициализируется, что приводит к созданию объектов `UISearchBar` и `UIWebView`. Контроллер представлений назначается делегатом для панели поиска.
2. Когда пользователь предоставляет набор поисковых терминов, панель поиска уведомляет метод `searchBarSearchButtonClicked` своего делегата. Этот метод форматирует Google URL в соответствии с предоставленными терминами и создает объект `NSURLRequest`. Затем этот объект передается Web-представлению при вызове его метода `NSURLRequest`.
3. Внутренние структуры Web-представления прокрутки управляют всем процессом выборки и отображения Web-страницы, прокруткой и переходами по ссылкам.

Для дальнейшего изучения

Web-представления очень легко реализовать, а выглядят они впечатляюще. Прежде чем продолжить изучение, попробуйте выполнить следующее:

- попробуйте загрузить Web-содержимое в Web-представление с помощью метода `loadHTMLString`;
- добавьте контроллер навигации с панелью инструментов. На панель инструментов поместите функциональные кнопки перехода вперед, перехода назад, остановки и перезагрузки содержимого;
- замените `UISearchBar` классом `UINavigationBar`. Добавьте в качестве подчиненного представления объект `UITextField` и измените код так, чтобы он выполнял функцию адресной панели обозревателя Интернета;
- проверьте наличие прототипов UIWebView.h и UISearchBar.h. Вы найдете их в папке /Developer/Platforms/iPhoneOS.platform внутри каталога Headers библиотеки UI Kit.

Глава 11

Параметры приложения

Параметр (setting), или настройка (preference), для приложения включает в себя две составляющие: *ключ* (key) и *значение* (value). Ключ — это уникальное значение, определяющее параметр, к примеру, `musicVolume`. Значение — это значение, хранящееся в данном ключе. Работа с параметрами состоит в создании и изменении пар "ключ/значение" в рамках списков свойств.

Списки свойств (property lists) являются файлами в формате XML, которые хранят пары "ключ/значение", составляющие параметры приложения. Для разработчиков Apple в этом нет ничего нового, поскольку такая технология поддерживается большинством самых последних версий Mac OS X, а пары "ключ/значение" могут напрямую считываться из классов словарей (dictionaries) и писаться в них, не прибегая при этом к разбору файла.

Словари и списки свойств

Списки свойств (property lists) физически являются файлами, расположенными на диске, но в вашем приложении представлены в виде словаря с ключами и значениями. Класс `NSDictionary` предоставляет методы, необходимые для считывания и записи списков свойств с диска и на диск, и может создавать пары "ключ/значение" без необходимости реализации разработчиком каких-либо сложных функций разбора (parsing). Класс `NSMutableDictionary` создает поверх всего этого механизм для добавления и удаления отдельных пар "ключ/значение".

Создание словаря

Словари могут быть созданы несколькими способами. Самый простой путь — это использовать класс `NSMutableDictionary`:

```
NSMutableDictionary *dict = [ [ NSMutableDictionary alloc ] init ];
```

Управление ключами

После создания словаря с помощью метода `setValue` вы можете добавить в него отдельные пары "ключ/значение":

```
[ dict setValue: @"myValue" forKey: @"myKey" ];
```

Также с помощью метода `setObject` вы можете задать любой объект в качестве значения ключа:

```
[ dict setObject: myObject forKey: @"myKey" ];
```

Чтобы удалить пару "ключ/значение", воспользуйтесь методом `removeObjectForKey`:

```
[ dict removeObjectForKey: @"myKey" ];
```

Для удаления всех пар "ключ/значение" воспользуйтесь методом `removeAllObjects`:

```
[ dict removeAllObjects ];
```

Запись списка свойств

Списки свойств записываются на диск, когда вы вызываете метод `writeToFile` словаря. Класс словаря автоматически конвертирует его данные в формат XML и записывает их по указанному пути:

```
[ dict writeToFile: path atomically: YES ];
```

Если файл пишется *поэлементно*, то список свойств сначала пишется во временный файл, а затем, переименованный, записывается по указанному пути. Это удобно в тех случаях, когда другие части вашего приложения в этот же момент времени могут считывать список свойств, и поможет предотвратить крах приложения, который может произойти при загрузке другим потоком частично записанного списка свойств.

Считывание списков свойств

Чтобы считать список свойств, который вы ранее сохранили, класс `NSDictionary` предоставляет метод `dictionaryWithContentsOfFile`. Это позволяет вам загружать список свойств непосредственно в класс словаря, не зная формата файла свойств и без необходимости осуществлять разбор этого файла.

Чтобы считать файл списка свойств в объект словаря, создайте объект словаря с помощью метода `initWithContentsOfFile`:

```
NSString path = [ NSString stringWithFormat:
   @"%@/Library/Preferences/bookmarks.plist", NSHomeDirectory()
];
NSMutableDictionary *dict = [ [ NSMutableDictionary alloc ]
    initWithContentsOfFile: path
];
```

Если объект словаря уже существует в памяти, то для замены содержимого словаря содержимым данного файла воспользуйтесь методом `setDictionary`:

```
[ dict setDictionary: [NSDictionary dictionaryWithContentsOfFile: path]];
```

Теперь вы можете работать со словарем, используя уже знакомые вами методы.

Для дальнейшего изучения

- Взгляните на прототип NSDictionary.h. Вы найдете его в папке /System/Library/Frameworks/Foundation.framework/Headers.
- Возьмите пример ShootStuffUp из *главы 10* и напишите код для считывания и записи настроек этого приложения-примера на диск. Воспользуйтесь этим для автоматической настройки элементов управления при загрузке приложения-примера.

Блок настроек приложения

В *главе 10* вы узнали о том, как создавать таблицы настроек, чтобы позволить пользователям изменять настройки в режиме реального времени. Это идеально подходит для тех приложений, в которых выход из приложения для внесения изменений может оказаться проблематичным. Однако для боль-

шинства приложений изменения в глобальные настройки приложения могут (и должны) вноситься через приложение Settings, поставляемое с iPhone. Когда вы создаете блок управления настройками приложения (preferences bundle), значок вашего приложения появляется внизу приложения Settings, предоставляя вашим пользователям единое стандартное место для редактирования параметров приложения.

Xcode предоставляет шаблон для создания блоков настроек и дает вам некоторое количество образцов, достаточных для начала работы. Чтобы добавить пакет настроек в свое приложение, откройте проект в Xcode и в меню **File** выберите команду **New File**. Вас попросят выбрать тип файла. В группе **iPhone OS** выберите категорию **Settings**, а затем выберите файл **Settings Bundle**. Когда появится соответствующее окно, назовите файл Settings.bundle и добавьте его в ваш проект. В ваше приложение будет добавлен блок-образец с примерами параметров в виде текстового поля, переключателя и бегунка.

Чтобы отредактировать параметры в вашем блоке управления настройками, откройте файл Root.plist. Он находится под папкой Settings.bundle, которая теперь существует на боковой панели вашего проекта.

По умолчанию список свойств отображается в редакторе списка свойств, облегчая тем самым добавление новых пар "ключ/значение". Если вы хотите работать с необработанным содержимым файла, щелкните правой кнопкой мыши по файлу и в меню **Open As** выберите команду **Plain Text File**. В результате в текстовом редакторе откроется файл, в котором вы можете редактировать необработанный XML. Несмотря на то, что редактирование списка свойств в редакторе свойств Xcode является достаточно простой операцией, тем не менее, мы подробно разберем, как это происходит.

Добавление ключей

Каждая запись блока настроек представлена в словаре в разделе `PreferenceSpecifiers` списка свойств. Какие бы ни были элементы определены в этом массиве, все они будут отображены пользователю при редактировании настроек приложения. Каждая строка в массиве может определять новое поле или групповой разделитель (group separator) для отображаемой таблицы настроек.

Чтобы добавить новую строку, щелкните правой кнопкой мыши по массиву `PreferenceSpecifiers` и в появившемся контекстном меню выберите коман-

ду **Add Row**. Наверх массива будет добавлена новая строка. Щелкните правой кнопкой мыши по новому элементу и в появившемся меню **Value Type** выберите команду **Dictionary**. В результате вид элемента будет изменен из поля ввода строки в словарь, который вы можете использовать для описания новой ячейки. Иначе, вы можете выбрать существующий элемент и скопировать его. Для этого снова выделите массив `PreferenceSpecifiers`, а затем сделайте операцию вставки. В результате строка будет продублирована, предоставив вам для работы шаблон.

Групповые разделители

Групповые разделители (group separators) позволяют вам разделять ваши таблицы настроек на отдельные логические группы. Чтобы добавить групповой разделитель, добавьте в ваш словарь ключ `Type`. Для этого ключа задайте значение `PSGroupSpecifier`. Создайте второй ключ, `Title`, чье значение представляет имя, отображаемое в качестве метки группы. Обе строки должны иметь тип `String`:

```
<array>
    <dict>
        <key>Type</key>
        <string>PSGroupSpecifier</string>
        <key>Title</key>
        <string>Main Settings</string>
    </dict>
</array>
```

Текстовые поля

Текстовые поля (text fields) позволяют вам принимать от пользователя прямой ввод данных, например, имя пользователя или пароль. Чтобы добавить текстовое поле, добавьте к новому элементу словаря в массиве `PreferenceSpecifiers` приведенные далее ключи. Все значения ключей представлены в виде строк, если не указано иное:

- `Type` — тип; типом текстового поля является `PSTextFieldSpecifier`;
- `Title` — заголовок, отображаемый слева от поля, например, "Username";
- `Key` — имя ключа, представляющее действительное значение. Используется в вашем коде;

- `DefaultValue` — значение по умолчанию для поля. Для обозначения отсутствия значения можно указать пустое значение;
- `IsSecure` — булево поле, которое вы должны установить в `true`, если поле принимает секретный текст, например, пароль. Вот корректный синтаксис для объявления булева поля:

```
<key>IsSecure</key>
<true/>
```

- `KeyboardType` — тип клавиатуры, которая должна появиться, когда пользователь приступит к вводу данных в это поле. Поддерживаются следующие значения:
  - `Alphabet`;
  - `NumbersAndPunctuation`;
  - `NumberPad`;
  - `URL`;
  - `EmailAddress`;
- `AutocapitalizationType` позволяет вам определять, должно ли приложение Settings автоматически делать прописными первые буквы слов, вводимых пользователем в это поле, и если должно, то по каким правилам. Возможными значениями являются:
  - `None`;
  - `Word`;
  - `Sentences`;
  - `AllCharacters`;
- `AutoCorrectionType` позволяет вам определить, должно ли приложение Settings применять к вводимому в данное поле тексту автозамену, и если должно, то по каким правилам. Возможными значениями являются `Default`, `Yes` и `No`.

Завершенное текстовое поле может выглядеть так, как показано в приведенном далее примере:

```
<dict>
    <key>Type</key>
    <string>PSTextFieldSpecifier</string>
    <key>Title</key>
```

```
    <string>Username</string>
    <key>Key</key>
    <string>username_preference</string>
    <key>DefaultValue</key>
    <string></string>
    <key>IsSecure</key>
    <false/>
    <key>KeyboardType</key>
    <string>Alphabet</string>
    <key>AutocapitalizationType</key>
    <string>None</string>
    <key>AutocorrectionType</key>
    <string>No</string>
</dict>
```

Изменение значений переключателей

Изменение значений переключателей (switches) приводит к отображению в панели настроек `UISwitch` переключателей, которые пользователь может установить в положение `On` или `Off`. Это удобно при добавлении булевых настроек для активации или деактивации определенной функциональности. Чтобы добавить поле переключателя, добавьте к новому элементу словаря в массиве `PreferenceSpecifiers` приведенные далее ключи. Все значения ключей представлены в виде строк, если не указано иное:

- `Type` — тип; типом поля переключателя является `PSToggleSwitchSpecifier`;
- `Title` — заголовок, отображаемый слева от поля, например, "Extra Lives";
- `Key` — имя ключа, представляющее действительное значение; используется в вашем коде;
- `TrueValue` — значение, в которое устанавливается поле `Key`, когда переключатель установлен в положение `On`;
- `FalseValue` — значение, в которое устанавливается поле `Key`, когда переключатель установлен в положение `Off`;
- `DefaultValue` — булево значение, определяет положение переключателя по умолчанию. Оно должно быть установлено в `true`, чтобы по умолча-

нию переключатель находился в положении `On`, и в `false`, чтобы по умолчанию переключатель находился в положении `Off`. Вот соответствующий пример:

```
<key>DefaultValue</key>
<false/>
```

Завершенная запись для переключателя может выглядеть так:

```
<dict>
    <key>Type</key>
    <string>PSToggleSwitchSpecifier</string>
    <key>Title</key>
    <string>Extra Points</string>
    <key>Key</key>
    <string>extrapoints_preference</string>
    <key>DefaultValue</key>
    <false/>
    <key>TrueValue</key>
    <string>YES</string>
    <key>FalseValue</key>
    <string>NO</string>
</dict>
```

Полосы прокрутки

Записи для полос прокрутки (sliders) отображают в таблице настроек `UISlider`. Полосы прокрутки позволяют пользователю приблизительно задавать диапазон значений и полезны для визуального выбора уровня сложности в игре, установки уровня громкости или определения других подобных диапазонов значений. Чтобы добавить поле полосы прокрутки, добавьте к новому элементу словаря в массиве `PreferenceSpecifiers` приведенные далее ключи. Все значения ключей представлены в виде строк, если не указано иное.

Полосы прокрутки не используют заголовки (titles), поэтому, как правило, лучше хранить полосы прокрутки в отдельной группе, используя в качестве заголовка метку группы:

- `Type` — тип; типом многозначного поля является `PSMultiValueSpecifier`;
- `Key` — имя ключа, представляющее действительное значение; используется в вашем коде;

- `MinimumValue` — числовое поле, определяющее минимальное значение для бегунка полосы прокрутки. Вы можете задать его как поле типа `integer`, так и как поле типа `real`, в зависимости от того, как вы хотите представлять значения: как целочисленные или как числа с плавающей точкой. Вот соответствующий пример:

```
<key>MinimumValue</key>
<real>10.0</real>
```

- `MaximumValue` — числовое поле, определяющее максимальное значение для бегунка полосы прокрутки. Вы можете задать его как поле типа `integer`, так и как поле типа `real`, в зависимости от того, как вы хотите представлять значения: как целочисленные или как числа с плавающей точкой. И оно должно соответствовать типу, использованному в поле `MinimumValue`;
- `DefaultValue` — численное поле, задающее значение по умолчанию для бегунка полосы прокрутки. Вы можете задать его как поле типа `integer`, так и как поле типа `real`, и оно не должно соответствовать типам других полей;
- `MinimumValueImage`, `MaximumValueImage` — минимальное и максимальное значения; чтобы отобразить рядом с минимальным и/или максимальным значениями полосы прокрутки изображение, укажите имя файла изображения, расположенного в программной папке.

Завершенная полоса прокрутки может выглядеть так:

```
<dict>
    <key>Type</key>
    <string>PSSliderSpecifier</string>
    <key>Key</key>
    <string>musicvolume_preference</string>
    <key>DefaultValue</key>
    <real>5.0</real>
    <key>MinimumValue</key>
    <real>0.0</real>
    <key>MaximumValue</key>
    <real>10.0</real>
    <key>MinimumValueImage</key>
    <string>minvalue.png</string>
    <key>MaximumValueImage</key>
    <string>maxvalue.png</string>
</dict>
```

Многозначные поля

Эквивалентом раскрывающегося меню в приложении Settings является многозначное поле (multivalue field). Многозначные поля позволяют вам задавать целый ряд различных параметров, давая при этом пользователю возможность выбрать в списке один из них. Чтобы добавить многозначное значение, добавьте к новому элементу словаря в массиве `PreferenceSpecifiers` приведенные далее ключи (все значения ключей представлены в виде строк, если не указано иное):

- `Type` — тип; типом многозначного поля является `PSSliderSpecifier`;
- `Key` — имя ключа, представляющее действительное значение; используется в вашем коде;
- `Title` — заголовок, отображаемый слева от поля, например, "Difficulty";
- `Titles` — массив возможных вариантов выбора, отображаемых пользователю. Вот соответствующий пример:

```
<key>Titles</key>
<array>
  <string>Easy</string>
  <string>Medium</string>
  <string>Hard</string>
</array>
```

- `Values` — массив, содержащий действительные значения, поставленные в соответствие каждому заголовку. Это те значения, которые видит ваше приложение. Далее приведен соответствующий пример. Каждое значение сопоставлено заголовку, использованному в массиве `Titles`:

```
<key>Values</key>
<array>
  <string>1</string>
  <string>2</string>
  <string>3</string>
</array>
```

- `DefaultValue` — значение варианта выбора по умолчанию; оно должно быть сопоставлено элементу в массиве `Values`.

Завершенное многозначное поле может выглядеть так:

```
<dict>
    <key>Type</key>
    <string>PSMultiValueSpecifier</string>
```

```
    <key>Title</key>
    <string>Difficulty</string>
    <key>Key</key>
    <string>difficulty_preference</string>
    <key>Values</key>
    <array>
        <string>1</string>
        <string>2</string>
        <string>3</string>
    </array>
    <key>Titles</key>
    <array>
        <string>Easy</string>
        <string>Medium</string>
        <string>Hard</string>
    </array>
    <key>DefaultValue</key>
    <string>2</string>
</dict>
```

Дочерние панели

Блоки настроек приложения позволяют вам, помимо групп, задавать дочерние панели (child panes). Когда пользователь касается дочерней панели, приложение Settings прокручивает новую панель и предоставляет параметры этой панели. Кнопка **Back** автоматически отображается пользователю, позволяя ему возвращаться к вашей корневой панели.

Чтобы добавить дочернюю панель, задайте следующие строковые поля:

- `Type` — тип; типом поля для дочерней панели является `PSChildPaneSpecifier`;
- `Key` — уникальный ключ для дочерней панели;
- `Title` — заголовок дочерней панели, отображаемый пользователю; дочерняя панель отображается в виде кнопки со стрелкой раскрытия;
- `File` — имя другого списка свойств в Settings.bundle, содержащего дочернюю панель, которую вы хотите отобразить. Уберите в имени файла расширение plist.

Завершенная запись для дочерней панели может выглядеть так:

```
<dict>
    <key>Type</key>
    <string>PSChildPaneSpecifier</string>
    <key>Title</key>
    <string>Extended Preferences</string>
    <key>Key</key>
    <string>extended_preferences</string>
    <key>File</key>
    <string>Extended</string>
</dict>
```

Считывание значений блока настроек приложения

Множество настроек в вашем блоке хранятся в стандартном списке свойств в папке Library/Preferences песочницы вашего приложения. Чтобы загрузить значения по отдельности, для доступа к параметрам списка свойств воспользуйтесь классом `NSUserDefaults`:

```
NSString *difficulty = [ [ NSUserDefaults standardUserDefaults ]
    stringForKey:@"difficulty_preference" ];
```

Иначе вы можете загрузить список свойств, содержащий все настройки непосредственно в объект `NSDictionary` и работать с ним так, как вы узнали из этой главы:

```
NSDictionary *settings = [ NSUserDefaults standardUserDefaults
    dictionaryRepresentation ];
NSLog(@"Difficulty: %@\n", [ settings valueForKey:
      @"difficulty_preference" ]);
```

Для дальнейшего изучения

Теперь, когда вы познакомились с блоками настроек, попробуйте выполнить следующие упражнения.

- Возьмите пример ShootStuffUp из *главы 10* и конвертируйте его в блок настроек приложения.
- Проверьте наличие прототипа NSUserDefaults.h. Вы найдете его в папке /System/Library/Frameworks/Foundation.framework/Headers.

Глава 12

Cover Flow

В *главе 5* вы познакомились с библиотекой Quartz Core и узнали о том, как работать с Core Animation для создания великолепных 3D-преобразований. Все это является базисом технологии Cover Flow от Apple, которую вы можете встретить в приложении iPhone под названием iPod при переворачивании страниц в альбомах в альбомной ориентации. Apple приватизировала свой класс Cover Flow, поэтому он не может использоваться с SDK. В этой главе представлен совместимый с SDK способ реализации стиля Cover Flow, однако Apple все равно может запретить публикацию вашего приложения в своем интернет-магазине, если посчитает, что оно повторяет функциональность одного из собственных встроенных приложений Apple.

Лейтон Дункан (Layton Duncan) из "Polar Bear Farm", автор таких популярных приложений AppStore, как Search, Record, Telegram и др., первым написал код, иллюстрирующий iPhone-версию примера CovertFlow от Apple из примеров Xcode для настольных систем. Мы позаимствовали исходный пример Лейтона, почистили его и переписали в соответствии с санкционированными интерфейсами SDK, поэтому вы сможете создавать великолепные эффекты Cover Flow, не используя при этом никакого частного API.

Собственный класс Cover Flow от Apple для формирования своего содержимого использует частный объект UI Kit под названием `UICoverFlowLayer`. Этот класс приватизирован, поэтому мы не можем использовать его для создания совместимого с SDK примера (в противном случае ваше приложение будет нарушать соглашения с Apple). В результате поведение нашего примера будет очень схоже с собственными представлениями Cover Flow от Apple,

но не полностью им идентично. Небольшая модификация должна придать этому примеру то, что вы хотели бы добавить в ваше приложение.

Программирование Cover Flow в SDK: CovertFlow

В этом примере вы создадите подкласс `CFView` класса `UIScrollView`. Класс `CFView` инициализируется с набором изображений, представляющих обложки альбома (или любое другое содержимое, которое вы хотите представить). Этот класс используется Core Animation для создания раскладки в стиле Cover Flow на экране iPhone. При прокручивании с помощью вашего пальца класс `UIScrollView` вызывает метод `scrollViewDidScroll` своего делегата. Этот метод вычисляет, обложка какого альбома находится в центре экрана, и отображает ее (рис. 12.1).

Рис. 12.1. Пример CovertFlow

Класс `CFView` проектировался с учетом повторного использования. Чтобы реализовать это в вашем собственном коде, посмотрите на то, как класс контроллера представлений инициализирует и считывает его свойство `selectedCover`.

Вы можете скомпилировать это приложение, приведенное в листингах 12.1—12.5, с помощью SDK, создав проект приложения CovertFlow на базе представления (обратите внимание на букву "t" в слове "Covert"). Чтобы скомпилировать ваш проект, вам придется добавить в него Quartz Core framework. Если вы хотите увидеть, как создавать все эти объекты с нуля, то удалите код Interface Builder.

Листинг 12.1. Прототипы делегата приложения CovertFlow (CovertFlowAppDelegate.h)

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

@class CovertFlowViewController;

@interface CovertFlowAppDelegate : NSObject <UIApplicationDelegate> {
    UIWindow *window;
    CovertFlowViewController *viewController;
}

@property (nonatomic, retain) IBOutlet UIWindow *window;
@property (nonatomic, retain) IBOutlet CovertFlowViewController
  *viewController;

@end
```

Листинг 12.2. Делегат приложения CovertFlow (CovertFlowAppDelegate.m)

```
#import "CovertFlowAppDelegate.h"
#import "CovertFlowViewController.h"

@implementation CovertFlowAppDelegate
@synthesize window;
@synthesize viewController;

- (void)applicationDidFinishLaunching:(UIApplication *)application {
    CGRect screenBounds = [ [ UIScreen mainScreen ] bounds ];

    self.window = [ [ [ UIWindow alloc ] initWithFrame: screenBounds ]
      autorelease ];
    viewController = [ [ CovertFlowViewController alloc ] init ];
```

```
    [ window addSubview: viewController.view ];
    [ window makeKeyAndVisible ];
}

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

@end
```

Листинг 12.3. Прототипы контроллера представления CovertFlow (CovertFlowViewController.h)

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

/* Количество пикселов, прокручиваемых до того, как следующая обложка */
/* перейдет на передний план */
#define SCROLL_PIXELS 60.0

/* Размер каждой обложки */
#define COVER_WIDTH_HEIGHT 128.0

@interface CFView : UIScrollView <UIScrollViewDelegate>
{
    CAScrollLayer *cfIntLayer;
    NSMutableArray *_covers;
    NSTimer *timer;
    int selectedCover;
}
- (id) initWithFrame:(struct CGRect)frame covers:(NSMutableArray
    *)covers;
- (void)layoutLayer:(CAScrollLayer *)layer;

@property(nonatomic,getter=getSelectedCover) int selectedCover;

@end
```

```
@interface CovertFlowViewController : UIViewController {
    NSMutableArray *covers;
    CFView *covertFlowView;
}

@end
```

Листинг 12.4. Контроллер представлений CovertFlow (CovertFlowViewController.m)

```
#import "CovertFlowViewController.h"

@implementation CFView

- (id) initWithFrame:(struct CGRect)frame covers:(NSMutableArray *)covers
{
    self = [ super initWithFrame: frame ];

    if (self != nil) {
      _covers = covers;
      selectedCover = 0;

      self.showsVerticalScrollIndicator = YES;
      self.showsHorizontalScrollIndicator = NO;
      self.delegate = self;
      self.scrollsToTop = NO;
      self.bouncesZoom = NO;

      cfIntLayer = [ [ CAScrollLayer alloc ] init ];
      cfIntLayer.bounds = CGRectMake(0.0, 0.0, frame.size.width,
            frame.size.height + COVER_WIDTH_HEIGHT);
      cfIntLayer.position = CGPointMake(160.0, 304.0);
      cfIntLayer.frame = frame;

      for (int i = 0; i < [ _covers count ]; i++) {
        NSLog(@"Initializing cfIntLayer layer %d\n", i);
```

```
        UIImageView *background = [[[ UIImageView alloc] initWithImage:
           [ _covers objectAtIndex: i ] ] autorelease ];
        background.frame = CGRectMake(0.0, 0.0, COVER_WIDTH_HEIGHT,
            COVER_WIDTH_HEIGHT);
        [ cfIntLayer addSublayer: background.layer ];
      }

      self.contentSize = CGSizeMake(320.0, ((frame.size.height) +
             (SCROLL_PIXELS * ([ _covers count ] -1))));

      [ self.layer addSublayer: cfIntLayer ];
      [ self layoutLayer: cfIntLayer ];
    }
    return self;
}

- (void)scrollViewDidScroll:(UIScrollView *)scrollView {
    selectedCover = (int) roundf((self.contentOffset.y/SCROLL_PIXELS));
    if (selectedCover > [ _covers count ] -1) {
        selectedCover = [ _covers count ] - 1;
    }
    [ self layoutLayer: cfIntLayer ];
}

- (void)setSelectedCover:(int)index {
    if (index != selectedCover) {
      selectedCover = index;
      [ self layoutLayer: cfIntLayer ];
      self.contentOffset = CGPointMake(self.contentOffset.x,
                                       selectedCover * SCROLL_PIXELS);
    }
}

- (int) getSelectedCover {
    return selectedCover;
}
```

```
-(void) layoutLayer:(CAScrollLayer *)layer
{
    CALayer *sublayer;
    NSArray *array;
    size_t i, count;
    CGRect rect, cfImageRect;
    CGSize cellSize, spacing, margin, size;
    CATransform3D leftTransform, rightTransform, sublayerTransform;
    float zCenterPosition, zSidePosition;
    float sideSpacingFactor, rowScaleFactor;
    float angle = 1.39;
    int x;

    size = [ layer bounds ].size;

    zCenterPosition = 60; /* Координата Z выбранной обложки */
    zSidePosition = 0;    /* Координаты Z по умолчанию для остальных */
                          /* обложек */
    sideSpacingFactor = .85; /* На каком расстоянии друг от друга */
                             /* должны располагаться слайды обложек */
    rowScaleFactor = .55;    /* Расстояние между основной обложкой */
                             /* и боковыми обложками */

    leftTransform = CATransform3DMakeRotation(angle, -1, 0, 0);
    rightTransform = CATransform3DMakeRotation(-angle, -1, 0, 0);

    margin = CGSizeMake(5.0, 5.0);
    spacing = CGSizeMake(5.0, 5.0);
    cellSize = CGSizeMake (COVER_WIDTH_HEIGHT, COVER_WIDTH_HEIGHT);

    margin.width += (size.width - cellSize.width * [ _covers count ]
                     - spacing.width * ([ _covers 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 */
/* происходили 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 = CGPointZero;
    cfImageRect = rect;

    /* Основное положение */
    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 < selectedCover) /* Левая сторона */
    {
      rect.origin.y += cellSize.height * sideSpacingFactor
        * (float) (selectedCover - x - rowScaleFactor);
      sublayer.zPosition = zSidePosition - 2.0 * (selectedCover - x);
      sublayer.transform = leftTransform;
    }
```

```
        else if (x > selectedCover) /* Правая сторона */
        {
          rect.origin.y -= cellSize.height * sideSpacingFactor
            * (float) (x - selectedCover - rowScaleFactor);
          sublayer.zPosition = zSidePosition - 2.0 * (x - selectedCover);
          sublayer.transform = rightTransform;
        }
        else /* Выбранная обложка */
        {
          sublayer.transform = CATransform3DIdentity;
          sublayer.zPosition = zCenterPosition;

          /* Располагаем в центре прокручиваемого уровня */
          [ layer scrollToPoint: CGPointMake(0, rect.origin.y
              - (([ layer bounds ].size.height - cellSize.width)/2.0))];

          /* Располагаем прокручиваемый уровень в центре представления */
          layer.position = CGPointMake(160.0f, 240.0f +
                                       (selectedCover * SCROLL_PIXELS));
        }
        [ sublayer setFrame: rect ];
    }
    [ CATransaction commit ];
}
@end

@implementation CovertFlowViewController

- (id)init {
    self = [ super init ];
    if (self != nil) {
        covers = [ [ NSMutableArray alloc ] init ];

        for (int i = 1; i < 6; i++) {
          NSLog(@"Loading demo image %d\n", i);
          UIImage *image = [ [ UIImage alloc ] initWithData:
              [ NSData dataWithContentsOfURL:
```

```
                [ NSURL URLWithString: [ NSString stringWithFormat:
                    @"http://www.zdziarski.com/demo/%d.png", i ] ] ]
            ];
        [ covers addObject: image ];
    }
}
    return self;
}

- (void)loadView {
    [ super loadView ];

    covertFlowView = [ [ CFView alloc ] initWithFrame:
        self.view.frame
        covers: covers
    ];

    covertFlowView.selectedCover = 2;
    self.view = covertFlowView;
}

- (BOOL)shouldAutorotateToInterfaceOrientation:
    (UIInterfaceOrientation)interfaceOrientation
{
    return (interfaceOrientation == UIInterfaceOrientationPortrait);
}

- (void)didReceiveMemoryWarning {
    [ super didReceiveMemoryWarning ];
}

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

@end
```

Листинг 12.5. Функция `main` для CovertFlow (main.m)

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

int main(int argc, char *argv[]) {
    NSAutoreleasePool * pool = [[NSAutoreleasePool alloc] init];
    int retVal = UIApplicationMain(argc, argv, nil,
                                   @"CovertFlowAppDelegate");
    [pool release];
    return retVal;
}
```

Как это работает

Вот как работает пример CovertFlow:

1. При порождении приложения создаются окно и контроллер представлений, как и в любом другом приложении на базе представления.
2. Контроллер представлений создает экземпляр класса `CFView`, включенного в пример. Затем он загружает с Web-узла пять демонстрационных изображений и создает массив объектов `UIImage`. Этот массив передается собственному методу инициализации класса `CFView`.
3. Класс `CFView` инициализирует собственные свойства как класс `UIScrollView`. Затем он создает для каждого изображения подуровень. Подуровни преобразуются либо в боковое представление, либо во фронтальное представление.
4. Когда вы задаете свойство `selectedCover`, обложка преобразуется во фронтальное представление, а положение представления прокрутки центрирует выбранную обложку.
5. Когда пользователь прокручивает обложки с помощью своего пальца, вызывается метод `scrollViewDidScroll`. Он вычисляет положение прокручиваемого окна и устанавливает активную обложку исходя из того, куда прокрутил пользователь. Затем уровни анимируются, чтобы перелистать к соответствующей обложке.

Для дальнейшего изучения

Теперь, когда у вас есть представление о том, как управлять уровнями и анимациями, попробуйте выполнить несколько упражнений.

- Измените этот пример так, чтобы он автоматически прокручивал каждую обложку с помощью объекта `NSTimer`.
- Мы намеренно оставили в этом примере использование книжной ориентации. Воспользуйтесь теми знаниями, которые вы получили в этой книге, чтобы отображать таблицу обложек, если iPhone находится в книжной ориентации, и переключаться в представление Cover Flow, когда iPhone поворачивается в альбомную ориентацию.
- В заголовочных файлах вашего SDK проверьте наличие следующих прототипов: CAScrollLayer.h, CAAnimation.h и CATransform3D.h. Вы найдете их в папке /Developer/Platforms/iPhoneOS.platform внутри каталога Headers библиотеки Quartz Core.

Глава 13

Перелистывание страниц

Одной из наиболее эстетически привлекательных функциональных возможностей iPhone является его способность листать страницы, перелистывая (flicking) предыдущую страницу влево или вправо, открывая при этом следующую страницу. В *главе 10* вы узнали об элементах управления "страницы" (page controls) и представлениях прокрутки (scroll views). Собранные вместе эти классы могут обеспечить эффект перелистывания страниц в том же стиле, который можно увидеть на домашнем экране iPhone или при перелистывании фотографий.

Эта глава покажет способ реализации полнофункционального класса `PageScrollView`, который выполняет всю сложную работу, необходимую для создания подобного отображения. Класс `PageScrollView` создает область прокручиваемого содержимого, объединив ширину всех страниц, которые вы хотите отображать. Например, ширина экрана iPhone составляет 320 пикселов. Чтобы отобразить 10 страниц, вы создаете `UIScrollView` с размером содержимого в 3200 пикселов. Затем каждая страница "приклеивается" к другой части представления прокрутки. Используя свойство `pagingEnabled` класса `UIScrollView` каждый раздел `UIScrollView` разделяется для создания "страниц" по размеру экрана, которые поочередно сменяют друг друга, как в Springboard. По мере того как пользователь осуществляет перелистывание, представление прокрутки автоматически перескакивает к ближайшей странице в соответствии с тем, куда он перелистал. Для перелистывания страниц пользователь также может коснуться любого элемента управления страницами внизу экрана. Отображаемый номер страницы легко вычисляется путем

деления общей ширины всего прокручиваемого содержимого на ширину одной страницы.

Для приложений, использующих сотни страниц, у вас может не хватить памяти для выделения области прокрутки, вмещающей их все. Второй пример этой главы демонстрирует другой способ реализации класса `PageScrollView`, который использует область прокрутки размером всего в три страницы: левая, правая и центральная. По мере того как пользователь осуществляет перелистывание, невидимые части представления прокрутки (обычно левая и правая страницы) динамически сменяются, чтобы отображать содержимое смежных страниц. Тем самым бережется память и создается впечатление бесшовной прокрутки, содержащей все представления.

Пример перелистывания страниц: PageControl

Этот пример (рис. 13.1) реализует собственный класс `PageScrollView` для отображения пяти страниц содержимого, которое вы можете перелистывать, прокручивая влево и вправо. Представленный в этом примере класс `PageScrollView` является абсолютно автономным. Для каждой страницы он принимает `NSMutableArray`, содержащий объект, порожденный от класса `UIView`. Затем этот класс дает место `UIPageControl` и автоматически обновляет его, когда пользователь перелистывает страницы с помощью своего пальца. Для смены страницы пользователь также может коснуться любого элемента управления страницами. Как и в случае со всеми примерами в этой книге, класс `PageScrollView` не нарушает никаких правил SDK и использует только санкционированные интерфейсы. Также посредством протокола `PageScrollViewDelegate`, который уведомляет о смене страниц своего делегата, он демонстрирует то, как работает реализация протокола. Поскольку класс `UIScrollView` управляет всей анимацией прокрутки, то вам вообще не придется прибегать к использованию Core Animation.

Вы можете скомпилировать это приложение, приведенное в листингах 13.1—13.7, с помощью SDK, создав проект приложения PageControl на базе представления. Помимо этого вам придется добавить в ваш проект два новых файла: PageScrollView.h и PageScrollView.m. Для этого в Xcode в меню **File** выберите команду **New File**, затем в группе шаблонов Cocoa, расположенной под строкой **Mac OS X**, выберите класс **Objective-C**. Если вы хотите увидеть, как создавать все эти объекты с нуля, то удалите код Interface Builder.

Рис. 13.1. Пример PageControl

Листинг 13.1. Прототипы делегата приложения PageControl (PageControlAppDelegate.h)

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

@class PageControlViewController;

@interface PageControlAppDelegate : NSObject <UIApplicationDelegate> {
    UIWindow *window;
    PageControlViewController *viewController;
}

@property (nonatomic, retain) IBOutlet UIWindow *window;
@property (nonatomic, retain) IBOutlet PageControlViewController
  *viewController;

@end
```

Листинг 13.2. Делегат приложения PageControl (PageControlAppDelegate.m)

```
#import "PageControlAppDelegate.h"
#import "PageControlViewController.h"
```

```
@implementation PageControlAppDelegate

@synthesize window;
@synthesize viewController;

- (void)applicationDidFinishLaunching:(UIApplication *)application {
    CGRect screenBounds = [ [ UIScreen mainScreen ] bounds ];

    self.window = [ [ [ UIWindow alloc ] initWithFrame: screenBounds ]
        autorelease
    ];

    viewController = [ [ PageControlViewController alloc ] init ];

    [ window addSubview: viewController.view ];
    [ window makeKeyAndVisible ];
}

- (void)dealloc {
    [ viewController release ];
    [ window release ];
    [ super dealloc ];
}
@end
```

Листинг 13.3. Прототипы класса `PageScrollView` приложения PageControl (PageScrollView.h)

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

@interface PageScrollView : UIView <UIScrollViewDelegate> {
    UIScrollView *scrollView;
    UIPageControl *pageControl;

    CGRect _pageRegion, _controlRegion;
    NSMutableArray *_pages;
    id _delegate;
}
```

```
- (void)layoutViews;
- (void) notifyPageChange;

@property(nonatomic,assign,getter=getPages) NSMutableArray * pages;
@property(nonatomic,assign,getter=getCurrentPage) int currentPage;
@property(nonatomic,assign,getter=getDelegate) id delegate;
@end

@protocol PageScrollViewDelegate<NSObject>

@optional

    -   (void) pageScrollViewDidChangeCurrentPage:
    (PageScrollView *)pageScrollView currentPage:(int)currentPage;
@end
```

Листинг 13.4. Класс `PageScrollView` приложения PageControl (PageScrollView.m)

```
#import "PageScrollView.h"

@implementation PageScrollView

- (id)initWithFrame:(CGRect)frame {
    self = [ super initWithFrame: frame ];
    if (self != nil) {
      _pages = nil;
      _pageRegion = CGRectMake(frame.origin.x, frame.origin.y,
          frame.size.width, frame.size.height - 60.0);
      _controlRegion = CGRectMake(frame.origin.x,
                                  frame.size.height - 60.0,
                                  frame.size.width, 60.0);
      self.delegate = nil;

      scrollView = [ [ UIScrollView alloc ] initWithFrame: _pageRegion ];
      scrollView.pagingEnabled = YES;
```

```
      scrollView.delegate = self;
      [ self addSubview: scrollView ];

      pageControl = [ [ UIPageControl alloc ] initWithFrame:
                  _controlRegion ];
    [ pageControl addTarget: self action:
        @selector(pageControlDidChange:)
          forControlEvents: UIControlEventValueChanged ];
    [ self addSubview: pageControl ];
    }
    return self;
}

- (void)setPages:(NSMutableArray *)pages {
    if (_pages != nil) {
        for(int i=0;i<[_pages count];i++) {
            [ [ _pages objectAtIndex: i ] removeFromSuperview ];
        }
    }
    _pages = pages;
    scrollView.contentOffset = CGPointMake(0.0, 0.0);
    scrollView.contentSize = CGSizeMake(_pageRegion.size.width *
                                        [ _pages count ],
                                          _pageRegion.size.height);
    pageControl.numberOfPages = [ _pages count ];
    pageControl.currentPage = 0;
    [ self layoutViews ];
}

- (void)layoutViews {
    for(int i=0;i<[ _pages count];i++) {
        UIView *page = [ _pages objectAtIndex: i ];
        CGRect bounds = page.bounds;
        CGRect frame = CGRectMake(_pageRegion.size.width * i, 0.0,
            _pageRegion.size.width, _pageRegion.size.height);
        page.frame = frame;
```

```
        page.bounds = bounds;
        [ scrollView addSubview: page ];
    }
}

- (id)getDelegate {
    return _delegate;
}

- (void)setDelegate:(id)delegate {
    _delegate = delegate;
}

- (NSMutableArray *)getPages {
    return _pages;
}

- (void)setCurrentPage:(int)page {
    [ scrollView setContentOffset:
        CGPointMake(_pageRegion.size.width * page,
                    scrollView.contentOffset.y)
      animated: YES
    ];
    pageControl.currentPage = page;
}

- (int)getCurrentPage {
    return (int) (scrollView.contentOffset.x / _pageRegion.size.width);
}

- (void)scrollViewDidEndDecelerating:(UIScrollView *)scrollView {
    pageControl.currentPage = self.currentPage;
    [ self notifyPageChange ];
}

- (void) pageControlDidChange: (id)sender
{
    UIPageControl *control = (UIPageControl *) sender;
    if (control == pageControl) {
```

```
        self.currentPage = control.currentPage;
    }
    [ self notifyPageChange ];
}

- (void) notifyPageChange {
    if (self.delegate != nil) {
      if ([ _delegate
              conformsToProtocol:@protocol(PageScrollViewDelegate) ]) {
        if ([ _delegate respondsToSelector:
          @selector(pageScrollViewDidChangeCurrentPage:currentPage:)])
          {
            [ self.delegate pageScrollViewDidChangeCurrentPage:
              (PageScrollView *)self currentPage: self.currentPage
            ];
          }
        }
    }
}

@end
```

Листинг 13.5. Прототипы контроллера представлений PageControl (PageControlViewController.h)

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

@interface PageControlViewController : UIViewController <PageScrollView-
Delegate> {
    NSMutableArray *pages;
    PageScrollView *scrollView;
}

- (void) pageScrollViewDidChangeCurrentPage:(PageScrollView
   *)pageScrollView
 currentPage:(int)currentPage;

@end
```

Листинг 13.6. Контроллер представлений PageControl (PageControlViewController.m)

```
#import "PageControlViewController.h"

@implementation PageControlViewController

- (void)loadView {
    [ super loadView ];

    /* Загружаем демонстрационные изображения для страниц */
    pages = [ [ NSMutableArray alloc ] init ];
    for (int i = 0; i < 5; i++) {
        NSLog(@"Loading demo image %d\n", i);

        UIImage *background = [ [ UIImage alloc ] initWithData:
            [ NSData dataWithContentsOfURL: [ NSURL URLWithString:
              @"http://www.zdziarski.com/demo/black.png" ] ]
        ];

        UIImage *image = [ [ UIImage alloc ] initWithData:
            [ NSData dataWithContentsOfURL: [ NSURL URLWithString:
            [ NSString stringWithFormat:
                @"http://www.zdziarski.com/demo/%d.png", i+1 ] ] ]
        ];

        UIImageView *page = [ [ UIImageView alloc ] initWithFrame: [
            [ UIScreen mainScreen ] applicationFrame ]
        ];
        page.image = background;

        UIImageView *subview = [ [ UIImageView alloc ] initWithFrame: [
            [ UIScreen mainScreen ] applicationFrame ]
        ];
        subview.image = image;
        subview.bounds = CGRectMake(0.0, 0.0, image.size.width,
                                    image.size.height);
```

```
        [ page addSubview: subview ];
        [ pages addObject: page ];
    }

    scrollView = [ [ PageScrollView alloc ] initWithFrame:
                 self.view.frame ];
    scrollView.pages = pages;
    scrollView.delegate = self;
    self.view = scrollView;
}

- (void)didReceiveMemoryWarning {
    [ super didReceiveMemoryWarning ];
}

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

- (void) pageScrollViewDidChangeCurrentPage:
     (PageScrollView *)pageScrollView currentPage:(int)currentPage
{
    NSLog(@"I'm now on page %d\n", currentPage);
}
@end
```

Листинг 13.7. Функция `main` для PageControl (main.m)

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

int main(int argc, char *argv[]) {
    NSAutoreleasePool * pool = [ [ NSAutoreleasePool alloc ] init ];
    int retVal = UIApplicationMain(argc, argv, nil,
                                   @"PageControlAppDelegate");
    [ pool release ];
    return retVal;
}
```

Как это работает

Вот как работает пример PageControl:

1. При порождении приложения создаются окно и контроллер представлений, как и в любом другом приложении на базе представления.
2. Контроллер представлений создает экземпляр класса `PageScrollView`, включенного в этот пример. Затем он загружает с Web-узла пять демонстрационных изображений и создает массив объектов `UIImageView`. Класс `PageScrollView` принимает любые производные от класса `UIView`. Этот массив присваивается свойству `pages` класса `PageScrollView`.
3. Класс `PageScrollView` инициализирует `UIScrollView`, имеющий ширину в 1600 пикселов (5 страниц по 320 пикселов). Каждая страница добавляется с координатой *X*, кратной 320 пикселам (ширина экрана). Разрешается прокрутка страниц для автоматического перескакивания к соответствующей странице. Вниз экрана добавляется `UIPageControl`, а в представлении прокрутки отображается первая страница.
4. Когда пользователь осуществляет перелистывание или касается какого-либо элемента управления страницами, в зону видимости прокручивается новая страница, а делегат уведомляется о смене страниц.
5. Если свойство `pages` снова устанавливается, то представление прокрутки перенастраивает само себя на новый набор страниц.

Для дальнейшего изучения

Представления прокрутки являются достаточно универсальными инструментами и, как вы только что узнали, предоставляют впечатляющую функциональность. Попробуйте выполнить приведенные далее упражнения.

- Перенастройте представление прокрутки так, чтобы оно прокручивало страницы по вертикали, а не по горизонтали.
- Если вы этого еще не сделали, то в заголовочных файлах вашего SDK проверьте наличие прототипов UIScrollView.h, UIView.h и UIPageControl.h. Вы найдете их в папке /Developer/Platforms /iPhoneOS.platform внутри каталога Headers библиотеки UI Kit.

Класс *PageScrollView* для нескольких представлений

В предыдущем примере вы узнали, как работать с классом `UIScrollView`, область содержимого которого занимает все отображаемые им представления. Это хорошо работает в случае использования небольшого количества представлений, однако если вы перелистываете сотни фотографий, то этот способ потребует слишком много памяти. Преимущество использования предыдущего подхода состоит в том, что вы можете отображать индикатор горизонтальной прокрутки, который позволяет пользователю понимать, сколько объектов он перелистывает. Преимущество использования примера класса в этом разделе состоит в том, что вы можете отображать гораздо больше представлений.

Приводимый далее пример модифицирует `PageScrollView` для использования области содержимого размером всего лишь в три страницы. По мере того как пользователь перелистывает влево и вправо, страницы в представлении плавно заменяются новыми страницами. В результате мы получаем перелистывание с помощью пальца, которое выглядит точно так же, как в примере PageControl, однако может обрабатывать сотни страниц. Также было добавлено свойство для скрытия объекта `UIPageControl` внизу страницы. Для этого установите свойство `showsPageControl` в `NO`.

Данный класс определен в листингах 13.8 и 13.9. Чтобы использовать этот класс в примере PageControl, просто замените файлы этого класса приведенными далее.

Листинг 13.8. Прототипы PageScrollView (PageScrollView.h)

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

@interface PageScrollView : UIView <UIScrollViewDelegate> {
    UIScrollView *scrollView;
    UIPageControl *pageControl;

    CGRect _pageRegion, _controlRegion;
    NSMutableArray *_pages;
    id _delegate;
    BOOL _showsPageControl;
    int _zeroPage;
```

```
}
- (void)layoutViews;
- (void)layoutScroller;
- (void)notifyPageChange;

@property(nonatomic,assign,getter=getPages) NSMutableArray *pages;
@property(nonatomic,assign,getter=getCurrentPage) int currentPage;
@property(nonatomic,assign,getter=getDelegate) id delegate;
@property(nonatomic,assign,getter=getShowsPageControl) BOOL
  showsPageControl;
@end

@protocol PageScrollViewDelegate<NSObject>

@optional

- (void) pageScrollViewDidChangeCurrentPage:(PageScrollView *)
pageScrollView currentPage:(int)currentPage;

@end
```

Листинг 13.9. PageScrollView (PageScrollView.m)

```
#import "PageScrollView.h"

@implementation PageScrollView

- (id)initWithFrame:(CGRect)frame {
    self = [ super initWithFrame: frame ];
    if (self != nil) {
       _pages = nil;
       _zeroPage = 0;
       _pageRegion = CGRectMake(frame.origin.x, frame.origin.y,
           frame.size.width, frame.size.height - 60.0);
       _controlRegion = CGRectMake(frame.origin.x,
                                   frame.size.height - 60.0,
            frame.size.width, 60.0);
```

```
        self.delegate = nil;
        scrollView = [[UIScrollView alloc ] initWithFrame: _pageRegion];
        scrollView.pagingEnabled = YES;
        scrollView.delegate = self;
      [ self addSubview: scrollView ];

        pageControl = [[UIPageControl alloc] initWithFrame:
                        _controlRegion];
      [ pageControl addTarget: self action:
          @selector(pageControlDidChange:)
          forControlEvents: UIControlEventValueChanged ];
      [ self addSubview: pageControl ];
    }
    return self;
}

- (void)setPages:(NSMutableArray *)pages {
    if (pages != nil) {
      for (int i=0;i<[_pages count];i++) {
        [ [ _pages objectAtIndex: i ] removeFromSuperview ];
      }
    }
    _pages = pages;
    scrollView.contentOffset = CGPointMake(0.0, 0.0);
    if ([ _pages count] < 3) {
      scrollView.contentSize = CGSizeMake(_pageRegion.size.width *
        [ _pages count ], _pageRegion.size.height);
    } else {
        scrollView.contentSize = CGSizeMake(_pageRegion.size.width * 3,
            _pageRegion.size.height);
        scrollView.showsHorizontalScrollIndicator = NO;
    }
    pageControl.numberOfPages = [ _pages count ];
    pageControl.currentPage = 0;
     [ self layoutViews ];
}
```

```
- (void)layoutViews {
    if ([ _pages count ] <= 3) {
        for(int i=0;i<[ _pages count];i++) {
            UIView *page = [ _pages objectAtIndex: i ];
            CGRect bounds = page.bounds;
            CGRect frame = CGRectMake(_pageRegion.size.width * i, 0.0,
                _pageRegion.size.width, _pageRegion.size.height);
            page.frame = frame;
            page.bounds = bounds;
            [ scrollView addSubview: page ];
        }
        return;
    }
    /* Для 3-х и более представлений добавьте их все как скрытые, */
    /* расположив соответственно странице */
    for (int i=0; i<[ _pages count]; i++) {
        UIView *page = [ _pages objectAtIndex: i ];
        CGRect bounds = page.bounds;
        CGRect frame = CGRectMake(0.0, 0.0, _pageRegion.size.width,
                                            _pageRegion.size.height);
        page.frame = frame;
        page.bounds = bounds;
        page.hidden = YES;
        [ scrollView addSubview: page ];
    }
    [ self layoutScroller ];
}

- (void)layoutScroller {
    UIView *page;
    CGRect bounds, frame;
    int pageNum = [ self getCurrentPage ];

    if ([ _pages count ] <= 3)
        return;
```

```
NSLog(@"Laying out scroller for page %d\n", pageNum);
/* Левая граница */
if (pageNum == 0) {
    for (int i=0; i<3; i++) {
        page = [ _pages objectAtIndex: i ];
        bounds = page.bounds;
        frame = CGRectMake(_pageRegion.size.width * i, 0.0,
            _pageRegion.size.width, _pageRegion.size.height);
        NSLog(@"\tOffset for Page %d = %f\n", i, frame.origin.x);
        page.frame = frame;
        page.bounds = bounds;
        page.hidden = NO;
    }
    page = [ _pages objectAtIndex: 3 ];
    page.hidden = YES;
    _zeroPage = 0;
}

/* Правая граница */
else if (pageNum == [ _pages count ] -1) {
  for (int i=pageNum-2; i<=pageNum; i++) {
    page = [ _pages objectAtIndex: i ];
    bounds = page.bounds;
    frame = CGRectMake(_pageRegion.size.width * (2-(pageNum-i)), 0.0,
            _pageRegion.size.width, _pageRegion.size.height);
    NSLog(@"\tOffset for Page %d = %f\n", i, frame.origin.x);
    page.frame = frame;
    page.bounds = bounds;
    page.hidden = NO;
  }
  page = [ _pages objectAtIndex: [ _pages count ]-3 ];
  page.hidden = YES;
 _zeroPage = pageNum - 2;
}

/* Все промежуточные страницы */
else {
```

```
      for (int i=pageNum-1; i<=pageNum+1; i++) {
        page = [ _pages objectAtIndex: i ];
        bounds = page.bounds;
        frame = CGRectMake(_pageRegion.size.width * (i-(pageNum-1)), 0.0,
                _pageRegion.size.width, _pageRegion.size.height);
        NSLog(@"\tOffset for Page %d = %f\n", i, frame.origin.x);
        page.frame = frame;
        page.bounds = bounds;
        page.hidden = NO;
      }
      for (int i=0; i< [ _pages count ]; i++) {
        if (i < pageNum-1 || i > pageNum + 1) {
           page = [ _pages objectAtIndex: i ];
           page.hidden = YES;
        }
      }
      scrollView.contentOffset = CGPointMake(_pageRegion.size.width,
                                             0.0);
     _zeroPage = pageNum-1;
    }
}

- (id)getDelegate {
    return _delegate;
}

- (void)setDelegate:(id)delegate {
    _delegate = delegate;
}

- (BOOL)getShowsPageControl {
    return _showsPageControl;
}

- (void)setShowsPageControl:(BOOL)showsPageControl {
    _showsPageControl = showsPageControl;
    if (_showsPageControl == NO) {
```

```
        _pageRegion = CGRectMake(self.frame.origin.x,
                                 self.frame.origin.y,
            self.frame.size.width, self.frame.size.height);
        pageControl.hidden = YES;
        scrollView.frame = _pageRegion;
    } else {
        _pageRegion = CGRectMake(self.frame.origin.x,
                                 self.frame.origin.y,
            self.frame.size.width, self.frame.size.height - 60.0);
        pageControl.hidden = NO;
        scrollView.frame = _pageRegion;
    }
}

- (NSMutableArray *)getPages {
    return _pages;
}

- (void)setCurrentPage:(int)page {
    [ scrollView setContentOffset: CGPointMake(0.0, 0.0) ];
    _zeroPage = page;
    [ self layoutScroller ];
    pageControl.currentPage = page;
}

- (int)getCurrentPage {
    return (int) (scrollView.contentOffset.x / _pageRegion.size.width) +
_zeroPage;
}

- (void)scrollViewDidEndDecelerating:(UIScrollView *)scrollView {
    pageControl.currentPage = self.currentPage;
    [ self layoutScroller ];
    [ self notifyPageChange ];
}

- (void) pageControlDidChange: (id)sender
{
```

```
    UIPageControl *control = (UIPageControl *) sender;
    if (control == pageControl) {
      [ scrollView setContentOffset: CGPointMake
       (pageRegion.size.width * (control.currentPage - _zeroPage), 0.0)
          animated: YES
      ];
    }
}

- (void)scrollViewDidEndScrollingAnimation:(UIScrollView *)scrollView {
    [ self layoutScroller ];
    [ self notifyPageChange ];
}

- (void) notifyPageChange {
    if (self.delegate != nil) {
      if ([ _delegate
         conformsToProtocol:@protocol(PageScrollViewDelegate) ]) {
         if ([ _delegate respondsToSelector:
           @selector(pageScrollViewDidChangeCurrentPage:currentPage:)])
         {
           [ self.delegate pageScrollViewDidChangeCurrentPage:
              (PageScrollView *)self currentPage: self.currentPage
           ];
         }
      }
   }
}
@end
```

Как это работает

Вот как работает этот новый усовершенствованный класс `PageScrollView`:

1. Когда класс порождается, создается класс `UIScrollView`. Если прикреплено не более трех страниц, то представление прокрутки ведет себя точно так же, как в предыдущем примере: оно выделяет пространство для со-

держимого в представлении прокрутки для всех страниц. Если прикреплено более трех страниц, то пространство для содержимого представления прокрутки задается по ширине трех страниц. Это позволяет пользователю прокручивать влево или вправо страницы относительно центра. Затем добавляются, но скрываются все остальные представления.

2. После того как пользователь осуществит перелистывание, вызовется метод `layoutScroller`. Он определит, какую страницу выбрал пользователь, исходя из положения представления прокрутки и номера самой крайней левой страницы (`_zeroPage`). Затем метод восстанавливает начало координат текущей страницы и ее двух ближайших соседей, чтобы они были корректно ориентированы в представлении прокрутки. Поскольку пользователь может видеть только текущую страницу, то он не будет видеть смежные страницы, которые были заменены.

3. Если пользователь находится на самой крайней левой (первой) странице или самой крайней правой (последней) странице, то эта страница будет ориентирована на один конец представления прокрутки, чтобы пользователь не мог прокрутить за нее. Если пользователь находится на любой другой странице, то эта страница будет ориентирована по центру прокручиваемого представления, а ее левый и правый соседи будут расположены по другим сторонам. Затем в качестве номера самой левой страницы устанавливается нулевая страница, чтобы класс мог отслеживать то, на какой странице находится пользователь.

4. Если пользователь коснется какого-либо элемента управления страницами, то элемент управления страницами прокрутит к положению предыдущей или следующей страницы. Когда анимация прокрутки будет завершена, класс `UIScrollView` вызовет метод-делегат `scrollViewDidEndScrollingAnimation`. А он вызовет `layoutScroller`, который снова пройдет тот же самый процесс раскладки страниц.

Глава 14

Библиотека Media Player

Библиотека Media Player существенно облегчает добавление в ваше приложение возможностей воспроизведения видео и аудио. Контроллер видеопроигрывателя автоматически управляет изменениями ориентации и переходами между окнами для воспроизведения множества различных видеоформатов на экране iPhone. Единственный вызов видеоконтроллера для воспроизведения фильма автоматически отобразит проигрыватель и его элементы управления. Приложение может выбирать, какие элементы управления отображать, настраивать положение водеопроигрывателя и управлять функциями воспроизведения и остановки. Видеопроигрыватель поддерживает следующие форматы: MOV, MP4, M4V и 3GP, а также множество аудиоформатов.

Чтобы воспользоваться библиотекой Media Player, вам потребуется добавить ее в ваш проект Xcode. Для этого щелкните правой кнопкой мыши по папке Frameworks в вашем проекте, а затем в появившемся контекстном меню выберите команду **Add | Framework**. Перейдите к папке MediaPlayer.framework, а затем нажмите кнопку **Add**.

Контроллеры видеопроигрывателя

Класс `MPMoviePlayerController` инициализируется с объектом `NSURL`, с которым вы уже знакомы. Вы можете породить класс `NSURL`, чтобы ссылаться на локальный файл или URL удаленного Web-узла. Чтобы инициализировать кон-

троллер видеопроигрывателя, воспользуйтесь методом `initWithContentURL` класса и предоставьте объект `NSURL`. Вот соответствующий пример:

```
MPMoviePlayerController *moviePlayer = [[MPMoviePlayerController alloc]
    initWithContentURL: [ NSURL URLWithString: @"http:// ..." ] ];
```

Чтобы инициализировать видеопроигрыватель для воспроизведения локального файла, воспользуйтесь методом `fileURLWithPath` класса `NSURL`:

```
NSString *path = [NSString stringWithFormat: @"%@/Documents/movie.m4a",
    NSHomeDirectory()];

MPMoviePlayerController *moviePlayer = [[ MPMoviePlayerController alloc]
    initWithContentURL: [ NSURL fileURLWithPath: path ] ];
```

Свойства

После создания контроллера видеопроигрывателя вы можете задать несколько свойств.

Элементы управления

Вы можете определить конфигурацию элементов управления видеопроигрывателем, задав свойство `movieControlMode`:

```
moviePlayer.movieControlMode = MPMovieControlModeDefault;
```

Для определения этого свойства можно использовать значения из табл. 14.1.

Таблица 14.1

Режим	**Описание**
`MPMovieControlModeDefault`	Отображает элементы управления: воспроизведение (play)/приостановка (pause), уровень громкости (volume) и шакала времени (timeline)
`MPMovieControlModeVolumeOnly`	Отображает только элемент управления уровнем громкости (volume)
`MPMovieControlModeHidden`	Не отображает элементы управления

Форматное соотношение

Также можно настроить форматное соотношение (aspect ratio) видеофильма, задав свойство `scalingMode`:

```
moviePlayer.scalingMode = MPMovieScalingModeAspectFit;
```

Можно использовать следующие значения формата (табл. 14.2).

Таблица 14.2

Значение	Описание
`MPMovieScalingModeNone`	Не применять никакого масштабирования
`MPMovieScalingModeAspectFit`	Равномерно распределить по экрану
`MPMovieScalingModeAspectFill`	Равномерно распределить по всему экрану; возможно усечение
`MPMovieScalingModeFill`	Растянуть на весь экран, не соблюдать форматное соотношение

Цвет фона

Цвет фона используется, когда видеопроигрыватель переходит из режима воспроизведения или в этот режим, а также для заполнения пустого пространства, если видеофильм занимает не весь экран. По умолчанию цвет фона черный, но вы можете поменять его, установив свойство `backgroundColor` в объект `UIColor`:

```
moviePlayer.backgroundColor = [ UIColor whiteColor ];
```

Начало и остановка воспроизведения видеофильма

Чтобы воспроизвести видеофильм, вызовите метод `play` видеопроигрывателя:

```
[ moviePlayer play ];
```

Видеоконтроллер автоматически перенесет ваше текущее представление в видеопроигрыватель и начнет воспроизведение видеофильма.

Воспроизведение видеофильма остановится, когда пользователь коснется кнопки **Done**, или когда будет вызван метод `stop` контроллера:

```
[ moviePlayer stop ];
```

Когда видеофильм остановится, проигрыватель автоматически вернет предыдущее представление, которое отображало ваше приложение.

Уведомления

Ваше приложение может настроить уведомления так, чтобы отправлять их, когда видеопроигрыватель будет завершать загрузку содержимого, завершать воспроизведение или когда пользователь будет менять его форматное соотношение. Видеопроигрыватель отправляет события в центр уведомлений Cocoa, который вы можете настроить на передачу этих событий объекту в вашем приложении. Чтобы получать уведомления, воспользуйтесь классом `NSNotificationCenter` для добавления к видеопроигрывателю наблюдателя:

```
NSNotificationCenter *notificationCenter = [ NSNotificationCenter
defaultCenter ];
[ notificationCenter addObserver: self
    selector:@selector(moviePlayerPreloadDidFinish:)
    name: MPMoviePlayerContentPreloadDidFinishNotification
    object: moviePlayer
];
```

Уведомления отправляются вашему классу-делегату и заданному вами методу-получателю. Параметр `notification` позволяет вам узнавать, какое событие инициировало вызов метода-делегата:

```
- (void)moviePlayerPreloadDidFinish:(NSNotification*)notification
{
    NSLog(@"All my content are belong to me!\n");
}
```

Вы можете получать следующие уведомления:

- `MPMoviePlayerContentPreloadDidFinishNotification` отправляется, когда видеопроигрыватель завершает загрузку своего содержимого. Поскольку содержимое может воспроизводиться даже при частичной своей загрузке, то это уведомление может отправляться после того, как уже начнется воспроизведение;
- `MPMoviePlayerScalingModeDidChangeNotification` отправляется, когда пользователь меняет режим масштабирования видеофильма. Для переключения между полноэкранным и оконным режимами пользователю нужно коснуться значка масштабирования;

- `MPMoviePlayerPlaybackDidFinishNotification` отправляется, когда видеофильм завершит свое воспроизведение, или когда пользователь нажмет кнопку **Done**.

Для дальнейшего изучения

Видеопроигрыватели являются достаточно простыми объектами, поэтому мы позволим вам поработать с ними без каких-либо существенных подсказок.

- Возьмите любой из приведенных в этой книге примеров и добавьте в него класс `MPMoviePlayerController`. Вам не потребуется добавлять какие-либо переходы или специальный код, поскольку контроллер автоматически захватит управление экраном, когда вы начнете воспроизведение.
- В заголовочных файлах вашего SDK проверьте наличие прототипа MPMoviePlayerController.h. Вы найдете его в папке /Developer/Platforms/iPhoneOS.platform внутри каталога Headers библиотеки Media Player.
- В заголовочных файлах Mac OS X проверьте наличие прототипа NSNotification.h. Вы найдете его в папке /System/Library/Frameworks/Foundation.framework/Headers.

Предметный указатель

A

ABGroup, объект 286
ABPeoplePickerNavigationController, класс 297
ABPerson, объект 286
ABPersonViewController, класс 296
ABRecord, объект 286
ABRecordRef, объект 286
Accelerometer 303
Accuracy 278
Action sheets 61, 129
Activity indicators 302, 361
Address Book UI, библиотека 296
AddressBook.framework, библиотека 285
Alert sheet 127
Alert views 61, 128
Altitude 278
Animation type 118
Apple developer keys 18
Apple's developer program 18
Application badge 160
Application service 162
Aspect ratio 479
Audio queue 223
Audio service 220
Audio Toolbox, библиотека 203
AVAudioPlayer, класс 204, 205
AVFoundation, библиотека 203, 204

B

Badges 62
Bundle 12
Buttons 324

C

Callback function 228, 243
Categories 40
CFBundleDisplayName 16
CFBundleExecutable 16
CFBundleIconFile 16
CFBundleIdentifier 16
CFBundleName 16
CFFTP, интерфейс 271
CFHTTP, интефейс 269
CFNetwork, библиотека 251
CFReadStream, функция 260
CFSocket, объект 253
CFSocketContext, объект 259
CFSocketCreate, функция 253
CFView, класс 446
CFWriteStream, функция 263
CGColor, класс 90
CGPoint, структура 166
CGRect, структура 64, 167
CGSize, структура 166
Child panes 443
Client 252
CLLocationManager, объект 274
Controller 25
Controls 103, 301, 303
Core Graphics, библиотека 89, 166
Core Location 29, 273
Cover Flow 445

D

Data source 330, 346
Data views 51

Datagram 252
DatePickerMode 393
Delegate protocols 39
Destructive buttons 129
Developer certificate 21
Dictionary 294
Directory 12
Disclosures 145
Dynamic linking 30

F

Flicking 457
Frame 64

G

Group separators 437

I

Image picker 376
Image viewers 51
Index bar 349
Input object 52
Inspector 52
Interface Builder 45, 77
    удаление из проекта 57
iPhone SDK 17

K

Key 433

L

Label 144, 292
Latitude 277
Layer 187
Longitude 277

M

maximumDate 393
Media Player, библиотека 477
minimumDate 393
Mobile provisioning profile 21
Model 25
Model-View-Controller (MVP) 24
MPMoviePlayerController, класс 477
Multivalue field 442
Multivalue properties 291

N

Navigation bars 61, 97
Navigation controllers 61
Navigation-based application 25
NSDictionary, класс 433
NSMutableDictionary, класс 433
NSURL, класс 163

O

Objective-C 32

P

Page controls 327
PageScrollView, класс 457
Peer-to-peer networks 252
People picker 297
Person view 296
Picker 302, 382
Pinch 176
Preference bundle 330, 436

Preferences tables 301, 330
Program directory 12
Progress indicators 302
Progress indicators 361
Properties 38, 289
Property lists 433
Protocol 39
Prototype 28
Proximity sensor 303

Q

Quartz Core,
библиотека 117, 187

R

Raw data 244
Root view controller 98

S

Safari 163
Sandbox 12
Scroll views 303, 415
Section lists 301, 346
Segmented controls 103, 308
Server 252
Sheet 128
Sliders 314
Socket 252
Socket streams 260
Sound buffer 227, 241
Static linking 30
status bar 62
Switches 312
Synthesizing 46
System buttons 106

T

Tab bar application 26
Tab Bar Controller 53
Tab bars 302, 401
Table 138
Table view controller 62
Table views 62
Text fields 302, 437
Text views 51, 61
Timestamp 278
Timing 118
Transitions 117
Transmission Control Protocol (TCP) 253

U

UI Kit, библиотека 29, 59
UIActivityIndicatorView, класс 362
UIAlertView, класс 128
UIApplication, класс 59, 158
UIButton, класс 324
UIColor, класс 88
UIControl, класс 301, 303, 304
UIDatePicker, класс 392
UIEvent, объект 171
UIFont, класс 87
UIImage, класс 365
UIIMageView, класс 374
UIPickerView, класс 382
UIScrollView, класс 415, 446
UITabBar, класс 401
UITabBarController, класс 401
UITableView, класс 137
UITableViewCell, класс 140
UITableViewCell, объект 334
UITableViewController, класс 137, 138
UITextField, класс 317
UITextView, класс 84
UIToolbar, класс 105
UITouch, класс 169
UIView, класс 304

UIViewController, класс 74
UIWebView, класс 423
UIWindow, класс 63
User Datagram Protocol (UDP) 252
Utility application 26

V

Value 433
View 25, 46, 60
View controller 47, 60
View-based application 25
Volume control 230
VU meters 209

W

Web view 51, 303, 423
Web-представление 51, 303, 423
Window 46, 53, 60
Window transitions 61
Window-based application 26

А

Адресная книга 286
Акселерометр 303
Анимация уровней 192
Аудиоочередь 223
Аудиосервис 220

Б

Бейдж 62
    приложения 160
Блок настроек 330
Блок управления настройками
  приложения 436

В

Вибрация 250
Видеопроигрыватель 477
    форматное соотношение 479
Волюметр 209
Временная метка 278
Выборщик 302, 382
    даты и времени 392
    изображений 376
    контактов 297
Высота над уровнем моря 278

Г

Групповые разделители 437

Д

Датчик приближения 303
Дейтаграмма 252
Динамическое связывание 30
Долгота 277

З

Звук:
    воспроизведение 207
    запись 237
    измерение мощности 208

Звуковой буфер 227, 241
Значение 433

И

Импорты 35
Индексная панель 349
Индикатор активности 302, 361
Индикатор прогресса 302, 361
Интерфейс 35
Источник данных 330, 346

К

Категория 40
Клиент 252
Ключ 433
Ключи разработчика Apple 18
Кнопка:
  с изображением 105
  системная 106
  текстовая 105
  уничтожения 129
Контроллер 25
  корневого представления 98
  навигации 61, 98
  представлений 47, 60, 74
  табличного представления 62
Кросс-компилятор 17

Л

Лист 128
  действий 61, 129
  предупреждений 127

М

Менеджер Core Location 274
Метка 144, 292
Метод 35
Многозначное поле 442
Модель 25
Модель — Представление — Контроллер 24

Н

Навигационное приложение 25
Необработанные данные 244

О

Обратный вызов 258
Объект ввода 52
Одноранговое сетевое взаимодействие 252
Окно 46, 53, 60
Ориентация 78

П

Пакет 12
Панель:
  вкладок 302, 401
  дочерняя 443
  инструментов 105
  навигации 61, 97
Переход:
  анимация 117
  создание 118
Переходы окон 61
Песочница 12
Пинча 176
Подстановка 43
Поток:
  записи 263
  сокетов 260
  чтения 260
Представление 25, 46, 60
  данных 51
  для отображения контактов 296

предупреждений 61, 128
прокрутки 303, 415
табличное 62
текстовое 51, 61
Преобразования уровней 193
Приложение:
AVMeter 209
BigImage 419
BounceDemo 195
ControllerDemo 79
CovertFlow 446
EndWorld 131
FlipDemo 120
HelloView 65
HelloWorld 71
ImageFun 370
NosePicker 384
OpenGL ES 26
PageControl 458
PageDemo 109
PinchMe 176
ShootStuffUp 337
SourceReader 92
TabDemo 406
TableDemo 149, 350
TouchDemo 179
WebDemo 426
WhereYouAt 280
вибрация 250
выборщик Дня независимости 395
магнитофон 246
на базе представления 25
оконное 26
проигрыватель PCM 230
с панелью вкладок 26
сервер анекдотов 265
счетчик касаний 173
Программный каталог 12
Проигрыватель, звуковой 205
Просмотр изображений 51
Протокол 39
делегаты 39
Прототип 28
Профиль мобильной инициализации 21

Р

Раскрытия 145
Распределения во времени 118

С

Свойство 38, 289
многозначное 291
Сервер 252
Сервис приложения 162
Сертификат разработчика 21
Сеть одноранговая 252
Симулятор iPhone 19
Синтезирование 46
Словарь 294
Служебное приложение 26
Сокет 252
Списки разделов 301, 346, 350
Списки свойств 433
Статическое связывание 30
Строка состояния 62, 158

Т

Таблица 138
настроек 301, 330
Текст:
выравнивание 86
размер 87
редактирование 86
цвет 88
шрифт 87
Текстовое поле 302, 437
Тип анимации 118
Точность 278

У

Уровень 187
Уровень громкости 230

Ф

Фрейм 64
Функция обратного вызова 228, 243

Ш

Шаблон 47
Широта 277

Э

Элемент управления 103, 301, 303
кнопка 324
переключатель 312
полоса прокрутки 314
сегментированный 103, 308
страница 327
текстовое поле 317

Я

Ячейка таблицы 140