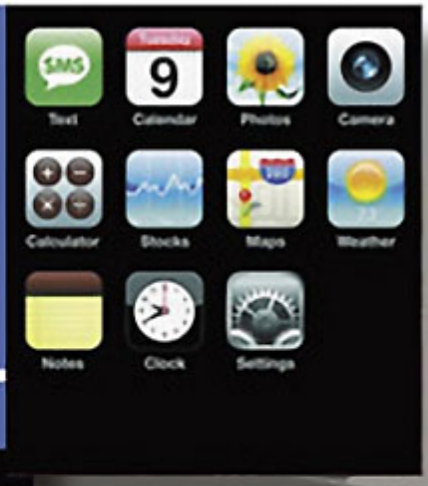
ДМИТРИЙ ЕЛИСЕЕВ

bhv®

Разработка и продажа программ для iPhone и iPad



- ОСНОВНЫЕ СВЕДЕНИЯ об Apple iOS
- РАЗРАБОТКА ПРОСТЕЙШЕЙ ПРОГРАММЫ В СРЕДЕ Xcode
- РАБОТА С GUI, ЗВУКОМ, ТАБЛИЦАМИ, ТЕКСТОВЫМИ И ГРАФИЧЕСКИМИ ДАННЫМИ
- ЗАГРУЗКА ДАННЫХ ИЗ ИНТЕРНЕТА
- БИБЛИОТЕКА ИГРОВОГО ФИЗИЧЕСКОГО МОДЕЛИРОВАНИЯ Box2D
- ГРАФИЧЕСКАЯ БИБЛИОТЕКА ДЛЯ СОЗДАНИЯ ИГР Cocos2D
- ОСНОВЫ ЯЗЫКА ObjectiveC

PRO
ПРОФЕССИОНАЛЬНОЕ ПРОГРАММИРОВАНИЕ

Дмитрий Елисеев

Разработка и продажа программ для iPhone и iPad

Санкт-Петербург

«БХВ-Петербург»

2011

УДК 681.3.06
ББК 32.973.26-018.2
Е59

Елисеев Д. В.

Е59 Разработка и продажа программ для iPhone и iPad. — СПб.: БХВ-Петербург, 2011. — 336 с.: ил. — (Профессиональное программирование)

ISBN 978-5-9775-0687-8

Рассмотрены вопросы создания программных приложений для мобильных устройств Apple iPhone и iPad. Изложены основные принципы функционирования программ в операционной системе Apple iOS, даны основы языка ObjectiveC. Рассмотрена работа с GUI, звуком, таблицами, текстовыми и графическими данными, приведены способы загрузки данных из Интернета.

Каждый раздел снабжен практическими примерами приложений. Описаны как разработка простейших программ в среде Xcode, так и создание сложных игровых программ с использованием профессиональных библиотек Box2D и Cocos2D. Особое внимание уделено практике размещения и продажи собственных программ в магазине Apple App Store.

Для программистов

УДК 681.3.06
ББК 32.973.26-018.2

Группа подготовки издания:

Главный редактор	*Екатерина Кондукова*
Зам. главного редактора	*Евгений Рыбаков*
Зав. редакцией	*Григорий Добин*
Редактор	*Елена Кашлакова*
Компьютерная верстка	*Натальи Караваевой*
Корректор	*Наталия Першакова*
Дизайн серии	*Инны Тачиной*
Оформление обложки	*Елены Беляевой*
Зав. производством	*Николай Тверских*

Лицензия ИД № 02429 от 24.07.00. Подписано в печать 22.07.11.
Формат 70×100 1/16. Печать офсетная. Усл. печ. л. 27,09.
Тираж 1500 экз. Заказ №
"БХВ-Петербург", 190005, Санкт-Петербург, Измайловский пр., 29.

Санитарно-эпидемиологическое заключение на продукцию
№ 77.99.60.953.Д.005770.05.09 от 26.05.2009 г. выдано Федеральной службой
по надзору в сфере защиты прав потребителей и благополучия человека.

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

ISBN 978-5-9775-0687-8

Оглавление

Введение

В настоящее время на рынке представлены сотни различных видов телефонов, смартфонов и карманных компьютеров. Но среди всего этого разнообразия есть одна компания, продукцию которой знают все, и как, конечно, догадались читатели, это фирма Apple. Это компания, поклонники которой разбивают палатки у магазина за неделю до начала продаж, чтобы стать первым обладателем новой модели устройства…

Конечно, основной успех Apple в настоящее время вызван линейкой телефонов iPhone, но даже без них имя Apple вписано в историю компьютерной техники: компания представила еще в 1977 г. персональный компьютер Apple-II и операционную систему Apple DOS (которая поддерживала длинные имена файлов, опередив на 15 лет MS Windows). Уже в 1984 г. Mac OS поддерживала оконный интерфейс и манипулятор "мышь", а в 1993 г. было выпущено устройство Apple MessagePad, представляющее собой карманный компьютер. Он имел процессор с тактовой частотой 20 МГц, 640 Кбайт памяти и экран с разрешением 240×336, управляемый стилусом. Пожалуй, MessagePad весьма опередил свое время, ведь аналогичное устройство от Palm появилось только через 3 года, а настоящий расцвет портативных устройств произошел, наверное, уже в XXI веке. Наконец, компания совершила переворот, представив смартфон iPhone, имеющий необычный (для 2007 г.) "бескнопочный" дизайн, экран, управляемый нажатием пальца, встроенный датчик ориентации устройства и прочие новшества, которые лишь потом стали спешно "клонироваться" в телефонах других производителей. И уже буквально во время написания этой книги Apple (в очередной раз) на шаг впереди конкурентов представила смартфон iPhone 4, не имеющий пока аналогов на рынке.

Но впрочем, эта книга не об истории, а о программировании. Поэтому стоит вернуться к настоящему. Сегодня смартфоны и планшетные компьютеры Apple пользуются заслуженной популярностью, поэтому изучение программирования для этой платформы может быть не только интересным, но и коммерчески привлекательным. *В главе 1*, от простого к сложному, описываются основные этапы создания программного обеспечения для устройств iPhone и iPad. Читатель найдет *в главах 2–5* информацию, необходимую для использования графического интерфейса операционной системы iOS, описание работы с файлами, звуком *в главах 6 и 7*, сетью Интернет. *В главе 8* рассматриваются практические примеры использования

библиотек Box2D и Cocos2D, применяющихся при создании игровых программ. Книга предоставляет информацию не только о программировании, *в главах 9–11* на практике описывается процесс продажи программ в магазине App Store, рассматриваются необходимые требования для выпуска программы в продажу, вплоть до открытия счета в банке.

Каждой теме посвящено отдельное приложение на сайте книги, что позволяет быстро найти нужную информацию или использовать фрагменты кода в качестве основы для новых проектов. Для компиляции примеров программ необходима среда разработки XCode 3.1, распространяемая бесплатно. Авторские примеры можно скачать по ссылке **http://www.bhv.ru/files/9785977506878.zip**.

Автор надеется, что книга позволит читателям подробнее изучить программирование для операционной системы iOS и, возможно, выпустить на рынок новые и интересные программы. Книга рассчитана на читателей, знакомых с основами программирования на языке C или C++.

Глава 1

Портативные устройства Apple

Смартфоны и портативные компьютеры Apple, имеющиеся на рынке

Компания Apple — одна из старейших компаний на компьютерном рынке, ее история начинается еще с 1976 г., когда Стив Джобс (Steve Jobs) и Стив Возняк (Steve Wozniak) создали один из первых персональных компьютеров. По одной из версий, название компании специально было подобрано так, чтобы быть в первых страницах телефонных справочников. С тех пор дела компании шли с переменным успехом: основными проблемами были закрытость архитектуры и использование процессоров Motorola, несовместимых с популярными IBM PC, и доля компьютеров Mac на рынке была не столь большой. Но в то же время она была достаточной и позволила компании развивать параллельные ниши, например плееры iPod и другие устройства.

Датой массового прихода Apple на рынок смартфонов можно считать 2007 г., когда первый iPhone был представлен на выставке MacWorld. Устройство объединило многие передовые тенденции, некоторые из которых были достаточно революционными: управление с помощью касания экрана пальцем, а не стилусом, поддержка технологии множественного касания Multi-touch, встроенный акселерометр и датчик ориентации с возможностью автоматического поворота интерфейса системы при изменении наклона устройства. И наконец, самое главное — уникальный дизайн, упрощенный до максимума: на устройстве только одна кнопка. В компании пришли к важному выводу: большинство пользователей покупают не "мегагерцы и мегабайты", им нужно красивое и эффектно выглядящее устройство. И расчет оправдался, смартфоны Apple не прошли незамеченными, и хотя одним такой концепт нравился, а другим — нет, равнодушных к новому устройству практически не осталось. По большому счету, это и есть основной фактор, определяющий популярность продукции у пользователей. В 2010 г. был выпущен новый смартфон iPhone 4, который опять-таки произвел фурор благодаря экрану сверхвысокого разрешения 960×640 и приятному дизайну. Ажиотаж был столь большим, что предварительные заказы размещались задолго до выхода.

С точки зрения программиста, все это приводит к простому выводу: разработка приложений для таких устройств не только интересна, но и перспективна и, возможно, прибыльна.

Все устройства Apple, работающие под управлением iOS, можно разделить на три категории:

- смартфоны Apple iPhone со встроенной операционной системой, дающей возможность пользователю загружать свои программы. Эти устройства имеют модули GSM и GPS, встроенную камеру и прочие атрибуты, свойственные современным телефонам;
- плееры iPod Touch — мало кто знает, что с точки зрения разработки программного обеспечения iPod Touch предоставляет практически те же возможности, что и iPhone, но при этом стоит вдвое дешевле;
- планшетные компьютеры Apple iPad с IPS-дисплеем диагональю 9,7 дюймов и разрешением 1024×768. iPad может выполнять также и приложения для iPhone, что позволяет успешно использовать его для отладки приложений, разрабатываемых как для iPhone, так и для iPad.

Итак, типы имеющихся устройств можно свести в табл. 1.1.

Таблица 1.1. *Устройства, работающие под управлением iOS*

Модель	Операционная система	Разрешение экрана	Процессор
iPhone 3G	2.0−4.2	480×320	412 МГц
iPhone 3GS	3.0−4.2	480×320	600 МГц
iPod Touch 3	3.0−4.2	480×320	600 МГц
iPhone 4	4.0−4.3	960×640	1 ГГц
iPod Touch 4	4.0−4.3	960×640	1 ГГц
iPad	3.2−4.3	1024×768	1 ГГц
iPad 2	4.3	1024×768	1 ГГц, два ядра

Приведенные данные тактовой частоты являются ориентировочными и взяты из сторонних источников, что, впрочем, для разработки прикладных программ (кроме игр) не так уж критично. Более важно то, что разрешение экрана новых смартфонов iPhone 4 ровно вдвое больше и обеспечивает совместимость со старыми версиями программ, ведь масштабирование в два раза может осуществляться системой быстро и без потери четкости.

Внешний вид iPhone 4 показан на рис. 1.1. В верхней части устройства имеются кнопка включения, разъем для наушников, на передней части можно видеть камеру для видеозвонков и кнопку **Home**, слева находятся кнопки регулировки громкости. В нижней части расположены динамик и разъем для подключения устройства к компьютеру. Через этот разъем iPhone подключается к компьютеру для зарядки и загрузки программного обеспечения (отладка программ также может проводиться на устройстве, а при его отсутствии — на симуляторе).

Рис. 1.1. Внешний вид Apple iPhone

Рис. 1.2. Внешний вид Apple iPad

Apple iPad выглядит практически так же, как увеличенный iPhone, это можно видеть на рис. 1.2. Операционная система обоих устройств одна и та же, что для разработчиков ПО является большим плюсом. Но, в отличие от iPhone, iPad не имеет камеры (она имеется только в новой модели iPad 2), в ряде моделей отсутствует GSM-модуль и приемник GPS. Процессор нового iPad 2 быстрее, чем в iPhone: очевидно, что более высокое разрешение экрана приводит к более высоким требованиям к памяти и быстродействию. На момент написания книги в продаже имеются модели с 16, 32 и 64 Гбайт флеш-памяти. Важно заметить, что память iPhone или iPad не может быть расширена, никакого слота для карт памяти там нет. Поэтому в отличие от программ, распространяемых на DVD, здесь пользователя вряд ли обрадует демонстрационный видеоролик HD-качества объемом в 100 Мбайт, вложенный в дистрибутив программы. С другой стороны, объема памяти предоставляемого устройствами обычно более чем достаточно для большинства прикладных программ.

Помимо "стандартных" устройств ввода (сенсорный экран, микрофон) iPhone и iPad имеют и специфические для этой платформы дополнительные средства — встроенные акселерометр и компас. С их помощью можно разнообразить управление программами (в основном конечно, игровыми, например, весьма популярны гонки, управление автомобилем в которых осуществляется с помощью наклона устройства).

Более-менее разобравшись с аппаратной частью, перейдем к программной: посмотрим, на каких элементах графического интерфейса построена операционная система Apple iOS, и как мы можем их использовать.

Основные принципы графического интерфейса Apple iOS

Несмотря на внешнее сходство с другими мобильными операционными системами, в Apple достаточно сильно переработан графический интерфейс. Основным для начинающего изучение iOS разработчика является документ "iOS Human Interface Guidelines", размещенный на сайте **http://developer.apple.com**, ознакомиться с которым следует каждому, кто собирается программировать или серьезно интересуется этой системой. В этом документе также содержится множество важных для разработки сведений, например размеры и форматы всех графических файлов, поставляемых с программой, без соблюдения которых программа не будет допущена к продаже в магазине App Store.

Наиболее важными для понимания являются несколько базовых принципов.

- Интерфейс iOS ориентирован на нажатие пальцем. Это обстоятельство накладывает ограничения на размер элементов, минимальный рекомендуемый Apple размер для комфортного нажатия составляет 44×44 пикселов для обычных экранов и 88×44 — для экранов высокого разрешения (960×640).
- Ориентация экрана может меняться в процессе работы программы. Два основных положения, портретное и ландшафтное, программа должна корректно масштабировать. К счастью для разработчиков, iOS предоставляет встроенные возможности, значительно облегчающие эту задачу.
- В один момент времени пользователь работает только с одной программой. В отличие от настольных компьютеров, в iOS нет многооконного интерфейса, и для небольших экранов это вполне оправданно. В то же время программы могут работать в фоне, начиная с версии 4.0, в iOS имеется многозадачность.
- Каждая программа имеет единственное "рабочее" окно. В iOS нет понятия многодокументного интерфейса (MDI, Multiple Document Interface), пользователь может работать лишь с одним окном в данный момент времени. Опять-таки, для небольших экранов это вполне логично и целесообразно.
- Интерфейс должен быть наглядным и интуитивно понятным. Для этого в iOS предусмотрены специальные средства: "переключатели", возможность изменения масштаба при помощи сжатия или растяжения объекта пальцами, перетаскивание объектов движением руки. Программные продукты в iOS приобретаются через сеть Интернет, обычно непосредственно с мобильного устройства, поэтому пользователь, скорее всего, не будет отдельно искать и загружать документацию. В идеале программа должна быть понятной пользователю без обращения к справке.

Вот некоторые рекомендации, которые дает Apple в документе "Human Interface Guidelines":

- Элементы управления должны выглядеть "нажимаемыми". Для встроенных компонентов iOS это уже реализовано, все кнопки и переключатели имеют контуры и градиенты, делающие их трехмерными.

- Структура программы должна быть понятной для навигации. Для этого iOS предоставляет встроенные средства, обеспечивающие удобное иерархическое представление, о них будет рассказано далее.
- Реакция программы на действия пользователя должна быть быстрой и понятной. Для этого iOS предоставляет встроенные средства, позволяющие сделать программу более красивой, такие как анимация или встроенные индикаторы прогресса. Эти "мелочи" позволяют пользоваться программой более комфортно.
- Зачастую пользователи мобильных устройств запускают приложения "на ходу", поэтому программа должна предоставлять доступ к содержимому просто и быстро. В этом плане логика работы пользователя за мобильным устройством отличается от действий за настольным компьютером.
- Желательно, чтобы программа сохраняла свое состояние при переключении. Новая версия iOS 4 поддерживает многозадачность, но есть еще немало пользователей предыдущих версий операционной системы. В них по нажатию кнопки **Home** программа закрывается, и это не должно приводить к потере важных для пользователя данных.
- В программе нет кнопки **Выход**. Это может показаться удивительным для тех, кто привык к Windows, тем не менее, программа в iOS не имеет кнопки закрытия. Для свертывания программы и появления главного экрана используется упомянутая кнопка **Home**.

Для иллюстрации этих принципов рассмотрим программу Mail, входящую в состав операционной системы Mac OS (рис. 1.3).

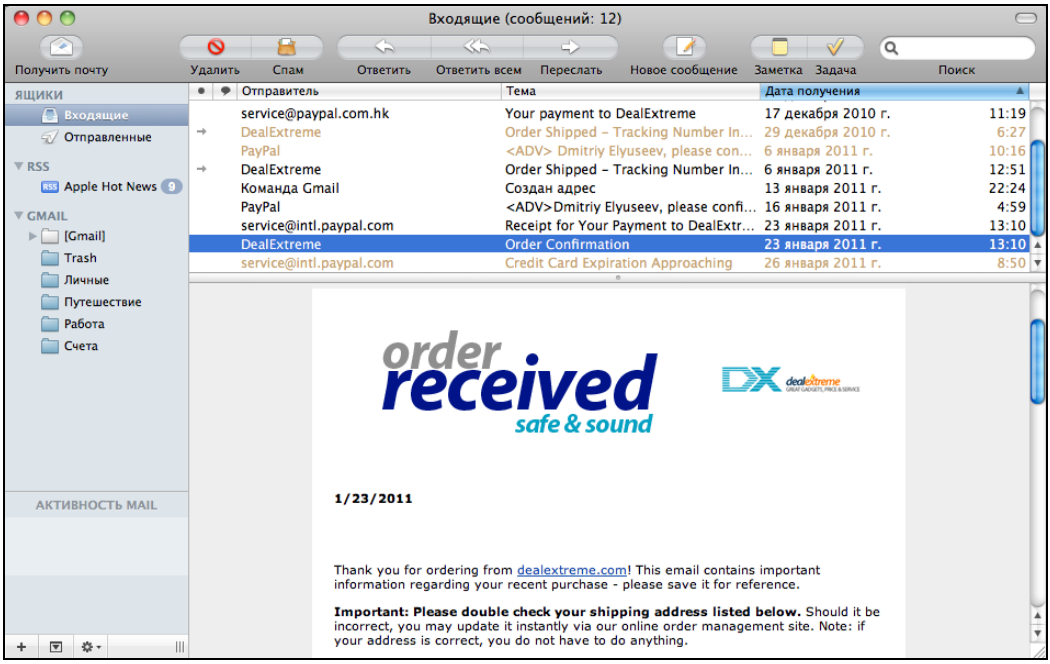


Рис. 1.3. Программа Mail для персонального компьютера

Здесь мы видим стандартное приложение для чтения электронной почты, однако в таком виде оно не поместится на экране мобильного телефона. Большое количество элементов управления, рассчитанных на использование "мыши", и несколько уровней вложенности интерфейса не могут быть с тем же удобством реализованы на экране меньшего размера. Для сравнения рассмотрим другую программу (окно программы настроек iPhone можно видеть на рис. 1.4).

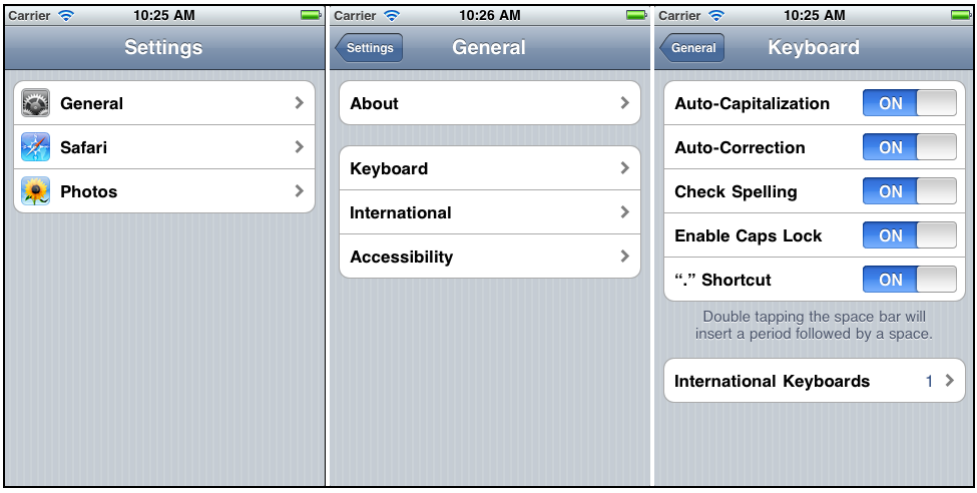


Рис. 1.4. Настройка параметров системы на iPhone

Вместо многооконного интерфейса с "параллельным" доступом (пользователь сразу может выбрать любую папку в почтовом ящике) был использован "последовательный", где пользователь открывает окна в четко заданной иерархии. Также можно видеть отсутствие кнопок маленького размера, ориентированных на "мышь", вместо них используются более крупные элементы управления.

На рис. 1.5 можно видеть ту же программу настроек, выполняющуюся на Apple iPad.

Рис. 1.5. Настройка параметров системы на iPad

Размер экранного пространства здесь позволил "сэкономить" один уровень вложенности и разместить на одном экране и список разделов, и параметры выбранного раздела. На рис. 1.3 можно заметить еще одну особенность — в iOS не используются древовидные списки. Действительно, сделать навигацию по дереву компонентов с помощью лишь нажатий пальцем было бы не очень удобно. Так что тем, кто привык пользоваться элементом `TreeCtrl` в Windows, придется привыкнуть к его отсутствию здесь. Раз уж речь зашла об элементах Windows, можно заметить, что в iOS также отсутствуют выпадающие списки `ComboBox`. Зато обычные линейные списки в iOS предоставляют гораздо больше возможностей.

Говоря об элементах управления, нельзя не упомянуть о "неэкранных" элементах iOS. Это, например, встряхивание устройства, обработчик которого пользователь может встроить в свою программу. Сложно придумать ему частое применение, в некоторых приложениях оно использовалось для обновления информации вместо кнопки **Refresh**. Впрочем, нельзя не заметить, что такое крупное устройство, как iPad, встряхивать не очень удобно, и подобным способом управления лучше не злоупотреблять.

Кроме встряхивания, пользователю доступны такие экзотические инструменты, как датчик положения, компас и акселерометр. Датчик положения может возвращать 6 значений: портретные и ландшафтные в двух ориентациях, а также два варианта горизонтального положения (экраном вверх и экраном вниз). Применение зависит от фантазии разработчика: например, если пользователь положил устройство на стол, можно отключить фоновый звук и приостановить игру. Подробнее об акселерометре будет рассказано далее, а пока вернемся к традиционным методам ввода и вывода информации. Рассмотрим основные элементы пользовательского интерфейса iOS более подробно.

Панель статуса (Status Bar)

В операционной системе iOS Status Bar находится в верхней части экрана. Панель статуса содержит информацию об устройстве и его окружении: уровень сигнала сети, заряд батареи, текущее время. Разработчик может указать в настройках программы, отображать панель статуса или работать в полноэкранном режиме. Но как рекомендует Apple, "нужно дважды подумать", прежде чем убрать Status Bar в своем приложении, ведь выводимая там информация является важной для пользователя (например, не видя уровня заряда батареи, пользователь может сильно разрядить свой смартфон и остаться без возможности совершить важный звонок).

Панель статуса можно видеть на рис. 1.4.

Панель навигации (Navigation Bar)

Панель навигации также можно видеть на рис. 1.4. Она всегда располагается в верхней части экрана под панелью статуса и, согласно своему названию, обеспечивает навигацию между различными окнами программы. Согласно рекомендациям Apple, надпись на панели навигации должна четко и ясно показывать пользователю его "местоположение" в иерархии интерфейса программы, надписи

должны быть легко читаемыми, их не рекомендуется закрывать дополнительными кнопками или картинками (не нужно забывать, что места на мобильных экранах довольно мало).

Панель инструментов (Toolbar)

В устройствах с iOS Toolbar располагается снизу. Toolbar может использоваться для выполнения каких-либо действий над содержимым текущего окна или для навигации, например содержать кнопки **Save** и **Cancel**, однако его нельзя использовать для переключения видов или вкладок — для этого имеется отдельный компонент Tab Bar. И как уже упоминалось, кнопки должны быть не меньше 44×44 пикселов для удобного нажатия пальцем.

Для выполнения стандартных действий (поиск, создание, удаление, воспроизведение) рекомендуется использовать стандартные значки, предоставляемые Apple, их описание можно найти в разделе документации "Standard Buttons for Use in Toolbars and Navigation Bars".

Пример использования панели инструментов также можно видеть на рис. 1.4.

Панель вкладок (Tab Bar)

Панель Tab Bar имеет такое же назначение, что и в Windows, вид панели вкладок в iOS показан на рис. 1.6.

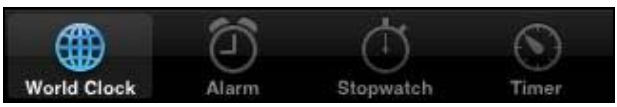


Рис. 1.6. Tab Bar в iOS

В отличие от Windows, панель вкладок всегда располагается в нижней части экрана. Согласно рекомендациям Apple, панель вкладок можно использовать только для переключения между разными окнами, относящимися к одному и тому же объекту. Нельзя использовать панель вкладок для выполнения каких-либо действий, для этого существует панель инструментов Toolbar, описанная ранее. Также не рекомендуется использовать более семи вкладок, это неудобно для восприятия (не говоря уже о том, что большое количество значков просто не поместится на экране iPhone).

Как и в панели инструментов, имеется большое количество системных значков, которые разработчик может использовать для стандартных функций, их можно найти в разделе документации "Standard Icons for Use in Tab Bars". В то же время Tab Bar (как, кстати, и Toolbar) позволяет использовать собственные значки, значки которых должны храниться в формате PNG с прозрачностью.

Всплывающее окно (Popover)

Вообще говоря, в iOS пользователю доступны только полноэкранные диалоговые окна, за исключением системных сообщений и индикаторов загрузки. Но для больших экранов iPad было сделано исключение: начиная с версии 3.2, появилась возможность вывода всплывающих окон, как показано на рис. 1.7.

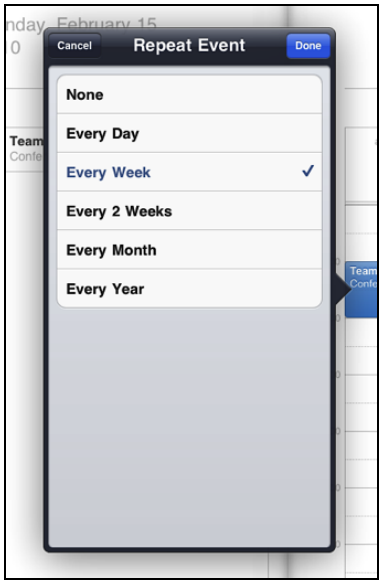


Рис. 1.7. Всплывающее окно в Apple iPad

В отличие от всплывающих окон в Windows, это окно имеет также стрелку, показывающую, откуда оно было открыто, что можно признать вполне удобным. В документации Apple всплывающие окна рекомендуют использовать для отображения более подробной информации в списках, дополнительных настроек и пр. Но при этом рекомендуется выводить окно так, чтобы оно не закрывало важной информации, особенно это касается вывода настроек (к примеру, довольно неудобно настраивать режим вывода изображения с помощью всплывающего окна, если окно настроек заслоняет то самое изображение, и результата не видно). Также согласно рекомендациям Apple, стрелка должна по возможности указывать непосредственно на элемент, к которому относится всплывающее окно (как показано на рис. 1.7).

Совмещенный вид (Split View)

Еще одним элементом управления, используемым исключительно на iPad, является Split View (см. рис. 1.5). С помощью Split View можно совместить два разнообразных элемента навигации, например список писем в левой панели и содержимое выбранного письма в правой, как это делается в популярных почтовых программах.

Таблица (Table View)

Все смартфоны имеют небольшой экран, поэтому основным способом отображения информации является список или таблица данных с возможностью прокрутки. Table View является, наверное, самым развитым из всех компонентов графического интерфейса, по возможностям заметно превосходя аналогичный элемент в Windows. Примеры табличного отображения показаны на рис. 1.8.

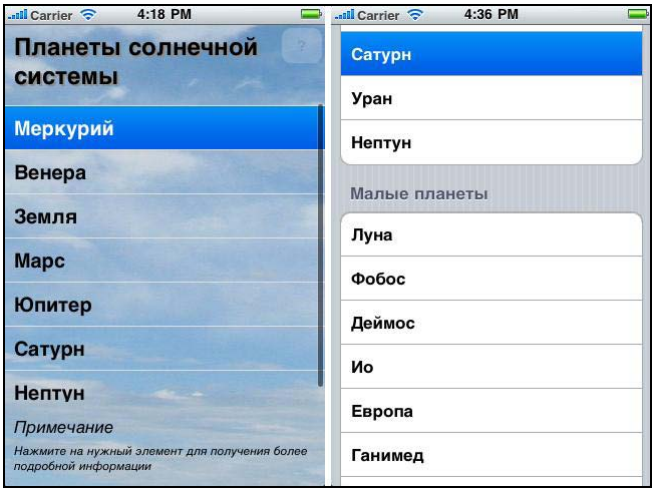


Рис. 1.8. Table View и различные режимы вывода

Как можно видеть, таблица может быть линейной, содержать справа колонку индексов, а также разбиваться на несколько сегментов. Помимо этого каждая ячейка в Table View может являться самостоятельным объектом со своим интерфейсом, что позволяет выводить сложно структурированные данные. Одним из примеров такого сложного табличного представления является показанный ранее на рис. 1.4 экран почтовой программы. Также Table View представляет встроенную поддержку анимации для скроллинга или удаления элементов.

Сообщение (Alert)

Вид окна сообщения показан на рис. 1.9.

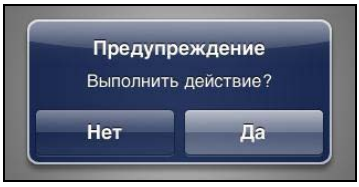


Рис. 1.9. Окно Alert в iOS

Интересно отметить прозрачную "подложку", сквозь которую видно содержимое окна нижнего уровня. Более того, в отличие от довольно-таки простой функции `MessageBox` в среде Windows, класс `UIAlertView` значительно более развит, он позволяет добавлять в сообщения не только несколько кнопок, но и другие элементы, например поле текстового ввода.

Окно действия (Action Sheet)

Это окно применяется для предоставления пользователю выбора из нескольких вариантов действий, что можно видеть на рис. 1.10.

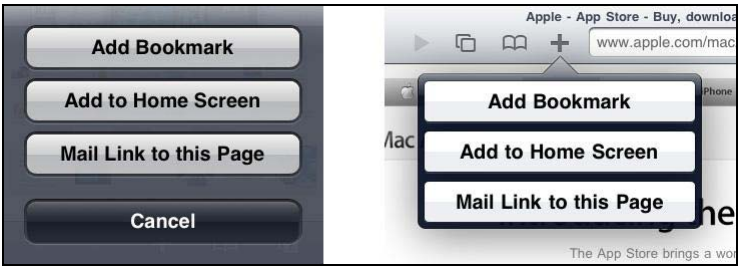


Рис. 1.10. Окно Action Sheet в iOS

На iPhone это окно автоматически располагается в нижней части экрана, на iPad оно открывается во всплывающем окне, что можно видеть на рисунке.

Отображение HTML (Web View)

Для разработчика важно то, что в Web View возможна загрузка не только страниц из сети Интернет с помощью ввода адреса, но также локально сохраненные или программно формируемые HTML-страницы, что позволяет использовать этот компонент для сложно форматируемого вывода, формирования отчетов, таблиц и пр. Подробнее об этом будет рассказано *в главе 4*.

Остальные элементы интерфейса

Текстовое поле ввода (Text View)

Стандартное поле ввода, по "последней моде" современных интерфейсов, имеет закругленные края. Через редактор ресурсов пользователь может настроить размер, тип шрифта и выравнивание, но сделать это можно для всего поля в целом, одновременное использование нескольких шрифтов в одном компоненте невозможно. Как и в других системах, пользователь может настроить вид текстового поля и тип используемых данных, например разрешить вводить только цифры, или только цифры и знаки препинания.

Помимо простого текстового поля, есть и модификации данного компонента. Например, текстовое поле поиска (Search Bar), представляющее собой поле ввода

с размещенными в нем значком и кнопкой **старт**, поле поиска с настраиваемыми параметрами (Scope Bar). Имеется также поле ввода для одиночной строки описания вводимого значения (Text Field).

Индикатор активности (Activity Indicator)

Поскольку в iOS курсора нет, этот элемент является статическим. Разработчик может поместить его в нужной части окна и делать видимым по мере необходимости: ожидание отображается в виде анимированного вращающегося круга.

Поле ввода даты и времени (Date and Time Picker)

Вид поля ввода даты в iOS показан на рис. 1.11. Как можно видеть, предлагаемый операционной системой компонент занимает весьма большое экранное пространство, поэтому зачастую целесообразнее использовать обычные текстовые поля для ввода даты или времени в цифровом формате.

Рис. 1.11. Окно **Date and Time Picker** в iOS

В iOS, помимо Date and Time Picker, существует и компонент Picker, позволяющий выбирать произвольные величины из списка.

Выбор из нескольких вариантов (Segmented Control)

Вид этого элемента управления показан на рис. 1.12.

Рис. 1.12. Окно **Segmented Control** в iOS

Он предоставляет пользователю возможность выбора только одного из вариантов.

Переключатель (Switch)

Вид этого компонента интерфейса представлен на рис. 1.13.

Рис. 1.13. Элемент управления Switch в iOS

Остальные элементы управления, такие как индикатор прогресса (Progress View), компоненты для скроллинга, вывода графических изображений, здесь не рассмотрены, их можно найти в редакторе ресурсов, предоставляемых средой разработки XCode или в документации на сайте Apple. Также разработчик может самостоятельно создавать элементы интерфейса.

Возможности и ограничения платформы Apple iOS

Самое время задаться вопросом о возможностях и ограничениях операционной системы для разработчика. Ключевым является принцип "песочницы" (Sandbox), по аналогии с детской игровой площадкой:

- программа может делать что угодно в пределах своей "песочницы";
- программа не может "залезть" в песочницу другой программы.

Мы можем работать с файлами, графикой, данными из сети Интернет, создавать модель нейронной сети или распознавать речь. Но мы не можем сохранить созданный нами файл в папку другой программы или обратиться напрямую к системным файлам. Хорошо это или плохо? Это хорошо, в первую очередь, тем, что защищает систему от вирусов и просто некачественных программ. Отсутствие полноценного межпрограммного взаимодействия вряд ли можно считать критичным для iOS, ведь не надо забывать, что это все-таки смартфон, а не полноценный компьютер. И вряд ли кто-либо будет использовать на смартфоне сложные программные комплексы. Что касается возможностей iOS для разработчика, то они весьма обширны. Разработчик может использовать в своих программах следующие технологии:

- операции чтения и записи данных в файлы и каталоги (но только в пределах собственной папки, как упоминалось ранее);
- операции работы с сетью как на высоком уровне (загрузка Web-страниц), так и на низком (sockets);
- 2D-графика посредством встроенных функций операционной системы;
- 2D- или 3D-графика посредством OpenGL ES, включая даже поддержку шейдеров;
- низкоуровневый доступ к процедурам записи и воспроизведения звука, позволяющий делать обработку аудиоданных или их визуализацию;
- ассемблерные вставки процессора ARM для более высокого быстродействия наиболее ответственных фрагментов программы;
- набор функций для аппаратной поддержки математических вычислений (таких как работа с векторами, преобразования Фурье), доступные в iOS 4.0 (accelerate framework).

Но в упомянутой бочке меда есть ложка дегтя, даже, скорее, две:

- сложность работы со сторонними файлами создает определенные неудобства для разработчиков, например пользователь не может штатными способами загрузить в программу файлы по беспроводной сети. Встроенная программа синхронизации Apple iTunes позволяет осуществлять синхронизацию только по кабелю, что является атавизмом в эпоху домашних и рабочих сетей Wi-Fi. По-

этому некоторые программы имеют свой встроенный Web-сервер, позволяющий пользователям загружать файлы через браузер с любого компьютера локальной сети. Пока что использование HTTP является практически единственным способом беспроводной коммуникации, хотя есть надежда, что здравый смысл восторжествует, и в новых версиях iTunes беспроводная синхронизация все же появится;

- операционная система iOS не позволяет запускать программы, не подписанные специальным сертификатом разработчика. Для его получения требуется членство в iPhone Developer Program, стоимость которого составляет 100 долларов в год. Таким образом, даже написав собственную программу, владелец iPhone не сможет просто взять и загрузить ее в устройство. Впрочем, в сети Интернет описываются и бесплатные способы создания сертификата для личного использования. Все программы можно без ограничений запускать на эмуляторе, который не требует сертификатов и распространяется бесплатно.

Конечно, эти ограничения являются неприятными, но далеко не фатальными. То, что ограничений для фантазии практически нет, можно видеть на примере Parrot AR Drone, описание которого можно найти на сайте **http://ardrone.parrot.com**.

Это полноценная летающая радиоуправляемая модель квадрокоптера, в качестве пульта к которой используется iPhone или iPad. Обмен данными осуществляется с помощью Wi-Fi, передатчик которой установлен на борту модели. Благодаря технологии Multi-touch владелец смартфона может управлять аппаратом, а встроенная видеокамера передает на iPhone изображение прямо с модели. Помимо стандартной программы управления полетом имеется также комплект средств для разработчиков, позволяющий, например, устраивать воздушные бои.

После рассмотрения подобного устройства самое время перейти к рассмотрению способов взаимодействия iOS с внешним миром.

Интерфейсы сети и внешних устройств

Как и любое современное устройство, iPhone и iPad имеют основные интерфейсы.

- Все устройства с iOS (iPhone, iPad и iPod Touch) имеют встроенный модуль Wi-Fi, позволяющий подключиться к беспроводной сети. Находящееся в сети устройство имеет собственный IP-адрес, что используется многими прикладными программами. Например, популярная программа Air Video просмотра видео в различных форматах состоит из сервера, формирующего видеопоток, и клиента на iPhone или iPad.
- Все смартфоны iPhone и некоторые модели iPad (с префиксом 3G) имеют GSM-модем. Знать, какой тип беспроводного соединения используется в конкретный момент, может быть полезно для экономии трафика, Apple предоставляет специальные классы, позволяющие учитывать это в прикладных программах.
- Все устройства имеют Bluetooth, однако его поддержка на уровне iOS ограничена. В то же время iOS имеет встроенный протокол GameKit, позволяющий на-

прямую соединять два устройства для передачи данных между ними. В Apple рекомендуют использовать GameKit для обмена короткими посылками данных объемом не более 1000 байт. Этого вполне достаточно, например, для несложных игр.

Другие интерфейсы

Помимо стандартных средств ввода-вывода, Apple стала одной из первых компаний, предоставившей пользователям новый уровень взаимодействия с портативным устройством. Например, наклон устройства или его встряхивание теперь могут использоваться для выполнения разнообразных действий в программе. Пользователю и разработчику iOS предлагает немало возможностей, способных значительно расширить функциональность смартфона.

Камера

Последние модели iPhone 4 снабжены камерой, оснащенной фотодиодной вспышкой, также возможна запись видео. Интересным приложением, например, для iPhone является программа чтения и распознавания штрихкодов.

К сожалению, iPad первой модели и ряд моделей iPod Touch не имеют камеры, поэтому разработчик должен предусмотреть проверку наличия камеры перед использованием соответствующих функций.

Акселерометр

Немаловажным устройством в iOS является акселерометр. Датчик измеряет наклон корпуса устройства относительно земли, и разработчик может учитывать эти данные в программе игр вроде гонок, когда наклон устройства используется для поворота руля или лабиринта, где нужно закатить шарик.

Рис. 1.14. Тестирование акселерометра при ходьбе с помощью AccelGraph

Частным случаем использования акселерометра являются изменение ориентации, которое автоматически отслеживается операционной системой, и встряхивание устройства. Для тестирования акселерометра есть удобная программа AccelGraph, которая распространяется бесплатно. Акселерометр можно использовать и достаточно необычным образом: например, с его помощью можно отслеживать даже сотрясение устройства во время ходьбы, что использовано в программе Шагомер.

Как показало тестирование в программе AccelGraph, ходьба со смартфоном действительно хорошо видна на графиках, что можно видеть на рис. 1.14.

Трехосевой гироскоп

Одним из нововведений в iPhone 4 является трехосевой гироскоп. Обычный акселерометр измеряет лишь углы наклона устройства относительно земли и поэтому не может реагировать на определенные виды движения, например на поворот игрока вокруг своей оси. Гироскоп в отличие от акселерометра, измеряет *угловое ускорение* относительно самого устройства, что дает еще одну степень свободы. Например, теперь можно поворачиваться в разные стороны вместе со смартфоном, и виртуальная камера будет поворачиваться в соответствующем направлении. Пример такой реализации можно посмотреть в игре "Eliminate Gun Range", сделанной специально для iPhone 4. Эта технология также может использоваться в проектах "дополненной реальности", позволяющих смартфону отображать объекты, "наложенные" на окружающую среду. Например, при наведении камеры смартфона на дом на экране устройства отобразится список компаний, находящихся в этом доме. Пока что эта технология еще развивается, в качестве одного из примеров реализации можно назвать бесплатную программу Layar, загрузить которую можно с сайта **http://www.layar.com**.

Компас

Еще одним датчиком для связи с внешним миром является компас. В отличие от акселерометра, ему сложно найти какое-либо реально полезное применение (за исключением программ навигации). К тому же, по личному опыту автора, работа компаса довольно неустойчива, стрелка часто колеблется и показывает не туда, в общем, обычный компас со своей задачей показывать направление на север справляется заметно лучше. Хотя для навигационных программ использование компаса для ориентации карты может быть весьма актуально.

GPS

Технология GPS давно уже стала привычной, тем не менее, без упоминания о ней этот раздел был бы неполным. Новые версии iPhone имеют встроенный блок GPS, а вот iPad имеет GPS-модуль лишь в 3G-версии.

В iOS встроен механизм определения координат даже на устройствах без модуля GPS — определение положения по IP-адресу с точностью до дома (естественно, при подключении к сети Интернет через домашнюю, а не сотовую сеть).

На этом мы закончим теоретическое введение и перейдем к практической части — изучению программирования для iOS.

Глава 2

Программирование для iPhone и iPad — первые шаги

Краткая информация об операционной системе Mac OS

Причем тут Mac OS, может подумать читатель, если книга описывает программирование для iPhone. На эту тему у автора для читателя есть две новости, одна плохая, одна хорошая. С какой начать? Плохая новость состоит в том, что среда разработки для iPhone работает в операционной системе Mac OS. Поэтому для освоения программирования iPhone придется изучить Mac OS, хотя бы на минимальном уровне, чтобы установить и запустить среду разработки XCode. Хорошая новость состоит в том, что благодаря программе эмуляции виртуальной машины VMWare установить Mac OS можно даже без покупки отдельного компьютера. В любом случае, познакомиться с этой системой придется, и это весьма интересно.

Mac OS была первой операционной системой, предоставляющей пользователям полноценный и дружественный графический интерфейс. Достаточно посмотреть на рис. 2.1: трудно поверить, что это было в 1984 году.

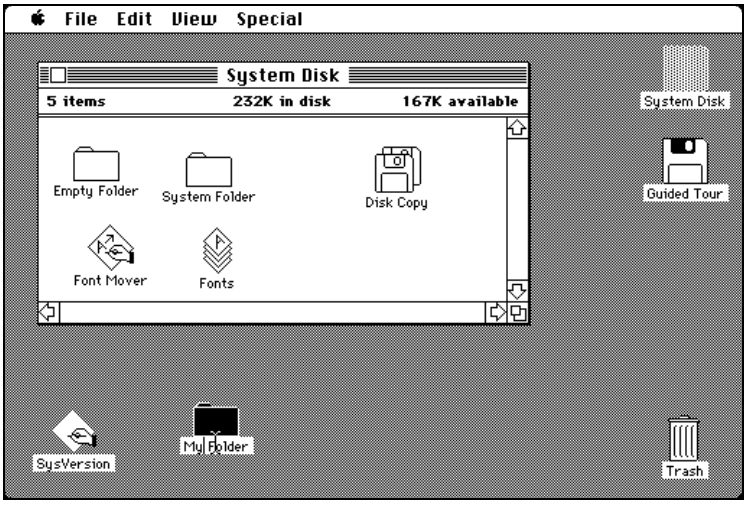


Рис. 2.1. Рабочий стол Mac OS, 1984 г., фото с сайта **http://ru.wikipedia.org**

Уже тогда в Mac OS была поддержка длинных имен файлов. В настоящее время трудно сказать, почему в итоге система Macintosh не стала популярной, да сейчас это уже и не столь актуально. Гораздо интереснее обратиться к системе Mac OS сегодняшнего дня. Здесь у пользователя есть три варианта действий.

- Установка Mac OS на виртуальной машине VMware. Самостоятельно настроить Mac OS для такой установки непросто, однако в сети Интернет можно найти уже готовый образ. Достаточно скачать и установить программу VMware Player, которая распространяется бесплатно — можно просто поискать в сети Интернет файл VMware-player-3.1.3-324285.exe, который и является инсталлятором этой программы. Размер образа Mac OS Snow Leopard.7z составляет около 10 Гбайт, для его работы требуется иметь компьютер с объемом памяти не менее 2 Гбайт и процессором не ниже, чем Core2Duo. В такой конфигурации скорость работы системы в виртуальной машине вполне достаточна для написания программ и подходит даже для разработки несложных игр.
- Установка Mac OS на обычный компьютер или ноутбук. Этот способ предпочтительнее первого по быстродействию, однако не так прост. В штатном варианте поставки система Mac OS привязана к BIOS компьютера производства Apple и на других системах просто не заработает. Существуют способы отключить эту привязку: в сети Интернет можно найти уже настроенные диски с системой, иронично называемой "Hakintosh". Выходом в этом плане может быть покупка компьютера такой же конфигурации, что и у оригинального Apple, в этом случае количество проблем минимально, а цена такого компьютера может быть в 1,5−2 раза меньше оригинального.
- Третий способ — просто пойти и купить компьютер или ноутбук Apple. Самым дешевым является Mac Mini. Этот настольный компьютер имеет небольшие размеры (вес около 1,5 кг) и достаточные для работы характеристики: процессор Core2Duo, 2 или 4 Гбайт памяти, видеокарта GeForce 320M. Ноутбуки MacBook стоят заметно дороже, да и клавиатура MacBook для пользователя персональных компьютеров несколько непривычна. Кстати, все компьютеры Macintosh позволяют устанавливать Windows в качестве второй операционной системы, так что купленный компьютер в любом случае пригодится, даже если не использовать Mac OS в дальнейшем.

Таким образом, существуют различные варианты действий на любой вкус и кошелек. Внешний вид операционной системы Mac OS показан на рис. 2.2.

Для тех, кто до этого использовал только Windows, полезно запомнить следующее:

- аналогом кнопки **Пуск** в Mac OS является яблоко в верхнем углу экрана;
- проводником в Mac OS является Finder;
- переключение раскладки между русским и английским языками выполняется комбинацией клавиш <Cmd>+<Пробел>;
- для работы с буфером обмена используются сочетания клавиш <Cmd>+<C> и <Cmd>+<V>;
- на клавиатуре нет отдельной клавиши <Delete>, ее заменяет сочетание клавиш <Fn>+<Backspace>;

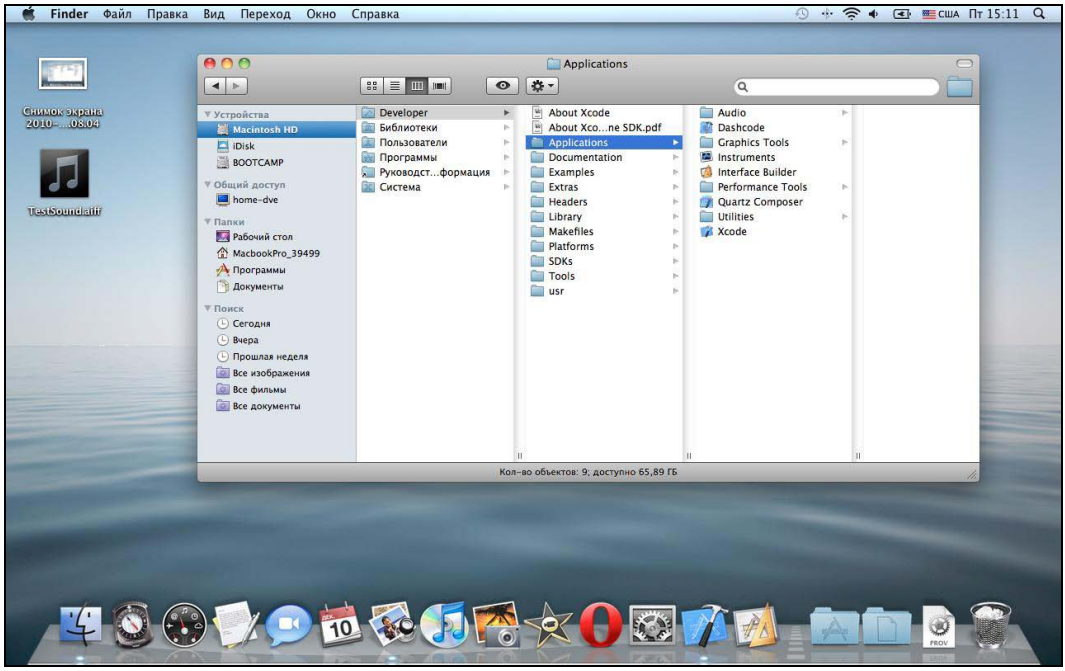


Рис. 2.2. Рабочий стол Mac OS версии Snow Leopard

- в качестве браузера по умолчанию используется Safari, хотя автору привычнее браузер Opera, который выглядит практически одинаково под Mac OS и Windows;
- для того чтобы сделать скриншот экрана, нужно нажать <Cmd>+<Shift>+<3>.

Да простят меня поклонники Mac за столь краткое изложение, но для первого знакомства с Mac OS перечисленного вполне достаточно. Интересующиеся более подробно устройством системы, могут обратиться к специальной литературе или online-документации.

Следующим необходимым шагом является установка среды разработки XCode. Ее установка бесплатна, достаточно регистрации на сайте **http://developer.apple.com**. Размер дистрибутива составляет около 2 Гбайт. Но перед тем как работать с XCode, настало время ознакомиться с основным языком программирования для iOS — Objective-C.

Основы языка Objective-C

Согласно сведениям из Википедии, язык Objective-C был создан в 1986 г. как объединение двух языков — C и Smalltalk. Язык Objective-C специально разрабатывался так, чтобы облегчить его изучение для C-программистов. И еще хорошая новость — язык Objective-C полностью совместим с языком C, т. е. все функции на языке C можно вставить в проект для iOS. Мы не будем вдаваться в теоретические особенности языка, а сразу перейдем к практике.

Объявление переменных

Как уже говорилось, мы можем создавать переменные как в C-стиле, так и в стиле Objective-C. Рассмотрим листинг 2.1.

Листинг 2.1. Объявление переменных

```
int i = 0;
float pi = 3.1415f;
int data1[4] = { 1, 2, 3, 4 }, data2[4] = {0};
BOOL res1 = TRUE;
Boolean res2 = YES, res3 = NO;
```

Логические переменные можно записывать как `Boolean` (ObjC-стиль), а можно как `BOOL` (WinAPI-стиль), кто к чему привык. Рассмотрим теперь более сложные случаи, которые пригодятся нам на практике (листинг 2.2).

Листинг 2.2. Объявление переменных (продолжение)

```
// прямоугольник
CGRect rect = CGRectMake(0, 0, 320, 480);
// точка
CGPoint point = CGPointMake(100, 100);
// строковые константы
NSString *str1 = @”This is a test string”, *str2 = @”100”;
// конвертация строки в число
int str2_val = [str2 intValue];
// конвертация числа в строку
NSString *pi = [NSString StringWithFormat:@”%f”, pi];
```

Структуры `CGRect` и `CGPoint` используются при работе с пользовательским интерфейсом. Например, чтобы сдвинуть кнопку или изображение в новое место, нужно изменить параметр `center`, который описывается как `GPoint`, а чтобы задать размеры, нужно изменить параметр `frame`, который описывается как `CGRect`.

Сами структуры описываются в листинге 2.3.

Листинг 2.3. Файл CGGeometry.h

```
/* Points. */

struct CGPoint {
  CGFloat x;
  CGFloat y;
};
```

```
typedef struct CGPoint CGPoint;

/* Sizes. */

struct CGSize {
  CGFloat width;
  CGFloat height;
};
typedef struct CGSize CGSize;

/* Rectangles. */

struct CGRect {
  CGPoint origin;
  CGSize size;
};
typedef struct CGRect CGRect;
```

Объявления похожи на структуры `RECT` и `POINT`, используемые в WinAPI, за исключением того, что для координат используются числа типа `float`.

> ***Примечание***
>
> Кстати, для любого элемента интерфейса, даже для кнопки, можно задать матрицу преобразования. Более подробно об этом будет рассказано далее.

Со строками ситуация чуть сложнее. Если `CGRect` и `CGPoint` — обычные структуры, то `NSString` — это класс, содержащий свои функции и имеющий разные методы инициализации (как метод `intValue`). О функциях в Objective-C стоит рассказать более подробно.

Функции в Objective-C

Поскольку Objective-C совместим с языком C, код листинга 2.4 корректен.

Листинг 2.4. Функции в C-стиле

```
int sum(int a, int b)
{
      return a+b;
}
// …
int a =10, b = 20;
NSString *s_sum = [NSString StringWithFormat:@"Сумма: %d", sum(a.b)];
```

Функции в стиле Objective-C описываются в листинге 2.5.

Листинг 2.5. Описание функций

```
// сумма 2 чисел
- (int) summa: (int)a: (int)b
{
      return a+b;
}
// преобразование строки в число
- (int) StrToInt: (NSString*)str
{
      return [str intValue];
}
```

Вызов функции в Objective-C осуществляется с помощью квадратных скобок. Две строки следующего кода делают одно и то же:

```
int i1 = 10, i2 = 20;
int res1 = sum(i1, i2), res2 = [self summa: i1: i2];
```

Как нетрудно догадаться, `self` является аналогом this в C++, он используется для вызова функций текущего класса. При вызове функций какого-либо объекта в квадратных скобках сначала ставится его имя, что можно видеть в приведенном примере `[str intValue]`. Здесь вызывается метод `intValue`, принадлежащий объекту `str`, что было бы эквивалентно записи `str.intValue()` в C++. Строка `[NSString StringWithFormat…]` означает вызов статической функции, что было бы эквивалентно следующей записи в C++: `NSString::StringWithFormat(…)`. Синтаксис отличается, а суть фактически остается той же самой.

Раз уж мы коснулись классов, пора перейти к их рассмотрению более подробно. Тем более что ни одна из современных систем программирования не обходится без них.

Классы в Objective-C

Поскольку речь пойдет об объектно-ориентированном программировании, нужно создать какой-нибудь полезный объект. Создадим класс для сравнения скорости эмулятора и реального устройства с iOS — метод с большим количеством вычислений, в качестве результата будет возвращаться время его работы. Итак, приступим (листинг 2.6).

Листинг 2.6. Класс измерения скорости работы

Файл SpeedCheck.h:

```
#import <Foundation/Foundation.h>
@interface SpeedCheck : NSObject
{
       // Время выполнения
       float fResults;
```

```
        // Время в миллисекундах
        int nResultsMS;
        // Результат математических операций
        float fCountData;
}

- (void) MakeCheck;
- (float) GetResults;

@property int nResultsMS;
@end
```

Файл SpeedCheck.m:

```
#import "SpeedCheck.h"

@implementation SpeedCheck
@synthesize nResultsMS;

-(id) init
{
        self = [super init];
        // Дополнительная инициализация (опционально)

        return self;
}

- (void) MakeCheck
{
        // Запоминание текущего времени
        NSDate *start = [NSDate date];

        // Выполнение вычислений
        float data = 0.0f;
        for(int i=0; i<4096; i++)
        {
                for(int j=0; j<4096; j++)
                        data += sin((float)j);
        }

        // Сохранение результатов
        fCountData = data;
        fResults = -[start timeIntervalSinceNow];

        self.nResultsMS = (int)(1000*fResults);
        // Отладка
        NSLog(@"Ready: %d, %d\n", nResultsMS, self.nResultsMS);
```

```
}

- (float) GetResults
{
       return fResults;
}

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

Сначала обратимся к определению класса — файлу SpeedCheck.h. Как можно видеть, `NSObject` — базовый класс в iOS, от которого наследуются все объекты. Директива `import` является аналогом директивы `include` в C. За ключевым словом `@interface`, внутри фигурных скобок, находятся переменные, хранящие данные объектов этого класса. Далее можно видеть описание функций-методов, которые в отличии от C++ описываются вне блоков фигурных скобок.

Ключевое слово `@property` служит для автоматического создания функций, обеспечивающих доступ к данным класса. Задавая какую-либо переменную как `property`, можно, например, определить доступ "только для чтения". Фактически указание свойства `property` является аналогом создания пары функций `get`/`set` для чтения и записи данных. Заканчивается описание класса ключевым словом `@end`.

Перейдем теперь к рассмотрению файла SpeedCheck.m. Как и в языке C++, первым идет включение заголовочного h-файла, только здесь используется описанная директива `import`. Следом можно видеть ключевое слово `@implementation`, указывающее, методы какого класса мы будем описывать, в одном файле может быть несколько классов. Как и в h-файле, заканчивается блок ключевым словом `@end`. Ключевое слово `@synthesize` указывает компилятору создать функции доступа для переменной `nResultsMS`, которую мы описали в h-файле.

Функции `init` и `dealloc` являются переопределенными функциями базового класса `NSObject`, можно видеть вызов функций базового класса с помощью ключевого слова `super`. В языке C++ аналогом строки `[super dealloc]` была бы строка `NSObject::dealloc`. Функция `init` выполняется во время инициализации класса, т. е. фактически является аналогом конструктора языка C++ (в ней можно произвести дополнительную инициализацию требуемых объектов). Функция `dealloc` вызывается, когда объект заканчивает существование. Обе эти функции не являются обязательными в наследуемых классах, здесь они показаны лишь для примера, никакого дополнительного кода здесь в них не выполняется.

Функция `MakeCheck` выполняет проверку скорости работы, ради чего этот класс и создавался. Она содержит вызов статической функции `date` системного класса `NSDate`. Эта функция возвращает текущую дату и время (сохраняются в объекте `start`), следующая функция `timeIntervalSinceNow` вызывается по окончании вычислений. Это уже не статическая функция, а функция объекта `start`, который был

проинициализирован временем начала вычислений. Соответственно результатом вызова `timeIntervalSinceNow` будет время с начала расчета, что нам и нужно. Время возвращается в секундах, соответственно для перевода в миллисекунды результат нужно умножить на 1000. Вызов функции `NSLog` используется для отладки, она позволяет выводить результаты в окно консоли, что удобно для просмотра и анализа значений переменных. По синтаксису эта функция аналогична функции `TRACE` в MFC или ATL. Функция `GetResults` возвращает время выполнения операции в секундах.

После того как класс создан, самое время его использовать. Но для этого нужно разобраться с управлением памятью в Objective-C, чем мы сейчас и займемся.

Управление памятью в Objective-C

Для языка Objective-C это, пожалуй, самая сложная и запутанная в понимании часть. В отличие от C++, где объект создается и удаляется одним-единственным способом (конструктор и деструктор), в Objective-C один и тот же объект может создаваться по-разному.

По умолчанию создание объектов происходит с помощью вызова функций `alloc` и `init` требуемого класса. Для нашего класса его создание будет выглядеть следующим образом:

```
SpeedCheck *check = [[SpeedCheck alloc] init];
```

Функция `alloc` создает объект, и уже для созданного объекта вызывается функция `init`. Когда объект не нужен, его можно удалить, вызвав функцию `release`.

```
[check release];
```

Очевидно, что если этого не сделать, будет утечка памяти. Таким образом, `alloc` и `release` являются аналогом `new` и `delete` в C++. Дальше начинаются особенности, которых нет в C++. Во-первых, этот код можно упростить, указав компилятору, что объект автоматически удалится по завершении функции, для этого используется ключевое слово `autorelease`:

```
SpeedCheck *check = [[[SpeedCheck alloc] init] autorelease];
```

Отдельный вызов функции `release` уже не нужен, это эквивалентно использованию smart pointers в языке C++.

В Objective-C имеется несколько правил управления памятью, которые важно запомнить. Чтобы лучше их понять, сначала рассмотрим несколько примеров из класса `NSString` в листинге 2.7.

Листинг 2.7. Разные методы класса `NSString`

```
// преобразование числа в строку
int nVal = 123;
NSString *string1 = [[NSString alloc] initWithFormat:@"%d", nVal];
// эта строка создана при помощи функции init***
// поэтому требует принудительного удаления
[string1 release];
// Эта строка не требует принудительного удаления
NSString *string2 = [NSString stringWithFormat:@"%d", nVal];
```

После этих примеров станут понятнее следующие правила:

- Мы можем удалять только те объекты, которые созданы нами с помощью функций, начинающихся со слов `alloc`, `init`, `copy` или `mutableCopy` (например, рассмотренная функция `initWithFormat` удовлетворяет этому условию).
- Если объект был создан с помощью другой функции (например, `stringWithFormat`), то мы не считаемся "собственниками" объекта, и удалять его нам не нужно, он удалится сам (обычно с помощью встроенной функции `autorelease`). Это удобно для временных объектов, но если нам нужно сохранить объект от удаления, мы можем "захватить" его с помощью функции `retain`. Эта функция увеличивает внутреннее число ссылок объекта, предотвращая его удаление.
- Когда объект не нужен, мы удаляем его с помощью вызова функции `release`.

Более подробную документацию можно найти в разделе "Object Ownership and Disposal" документа "Memory Management Programming Guide", размещенного на сайте **http://developer.apple.com**.

Теперь мы можем с окончательным пониманием сути дела полностью записать код, вызывающий созданный нами класс (листинг 2.8).

Листинг 2.8. Использование класса `SpeedCheck`

```
// Создание объекта
SpeedCheck *check = [[SpeedCheck alloc] init];
// Вызов метода, выполняющегося продолжительное время
[check MakeCheck];
// Получение значения с помощью Get-функции
float fres = [check GetResults];
// Получение значения с помощью property (альтернативный метод)
int   nres = check.nResultsMS;
// Удаление объекта
[check release];
```

На этом мы закончим краткое теоретическое введение в Objective-C и перейдем к практической части — среде разработки XCode.

Среда разработки XCode

Среда разработки XCode предоставляется Apple только в виде инсталлятора для операционной системы Mac OS. Кроме этого небольшого факта, никаких сложностей инсталляция XCode не представляет. Внешний вид интерфейса среды XCode показан на рис. 2.3.

Интерфейс включает в себя все основные возможности: дерево файлов проекта, подсветку синтаксиса, быстрый переход к нужной функции в файле и пр.

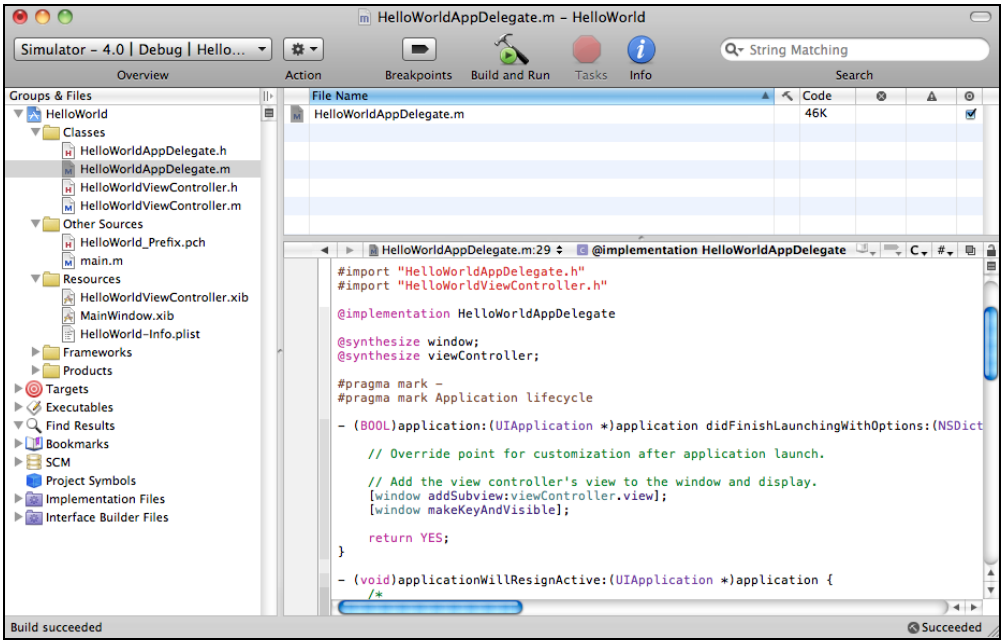


Рис. 2.3. Среда разработки XCode

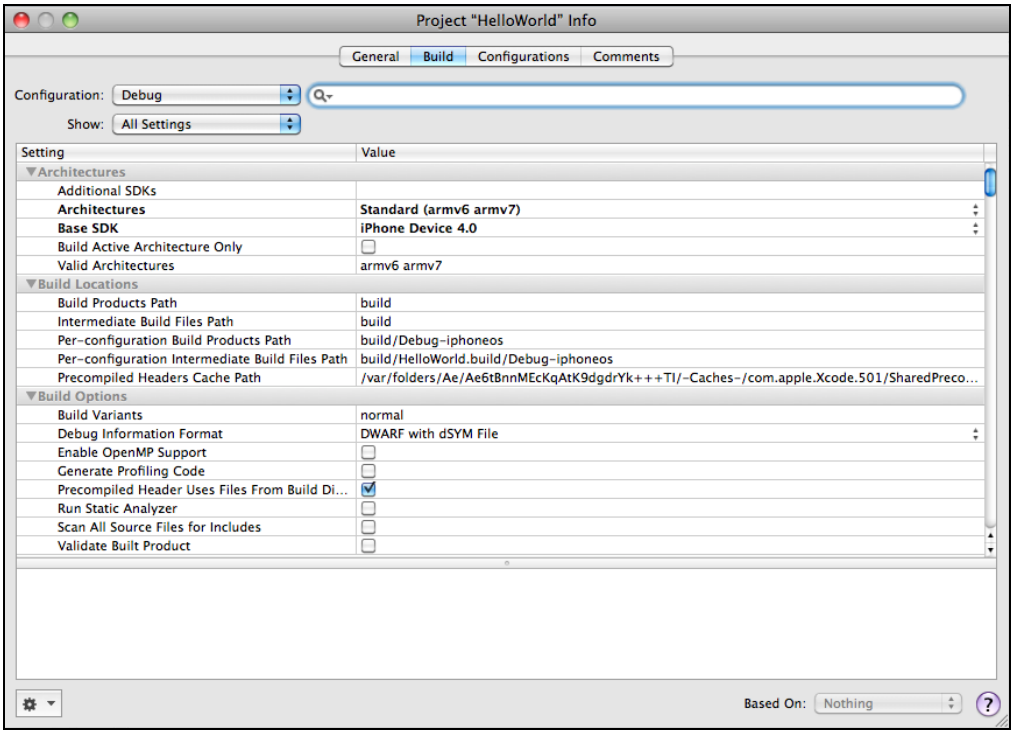


Рис. 2.4. Окно настроек проекта в XCode

Большая часть настроек производится через меню свойств проекта (рис. 2.4). Отметим некоторые из них своим вниманием:

- **iPhone OS Deployment Target** — минимальная версия операционной системы, необходимая для запуска программы. По умолчанию стоит **iPhone Device 4.0**, а достаточно большое количество пользователей еще могут использовать 3.0 или 3.1 (версия 4.0 за счет многозадачности более требовательна к быстродействию смартфона). Если программа не использует специфические особенности новых операционных систем, это значение лучше поменять, чтобы не сокращать зря количество потенциальных пользователей программы.
- Выбор устройств, на которых будет выполняться программа. Доступны три варианта: **iPhone**, **iPad** или **iPhone + iPad**. Если выбран тип **iPhone**, то программа будет запускаться и на iPad, но лишь в режиме 320×480. Если выбран тип **iPhone+iPad**, это может потребовать от разработчика дополнительных мер оптимизации интерфейса, чтобы он красиво выглядел при разных разрешениях и пропорциях экрана.

Осуществлять отладку программы можно и на реальном устройстве, однако иметь в наличии все устройства (iPhone, iPhone 4 и iPad) довольно затратно. Симулятор значительно облегчает процесс написания программы, к тому же избавляет от необходимости держать на столе лишние устройства и провода. Внешний вид симулятора показан на рис. 2.5.

Возможности симулятора практически полностью соответствуют реальному устройству, имеются функции разворота экрана, поддержка множественных касаний multi-touch (правда, более ограниченная).

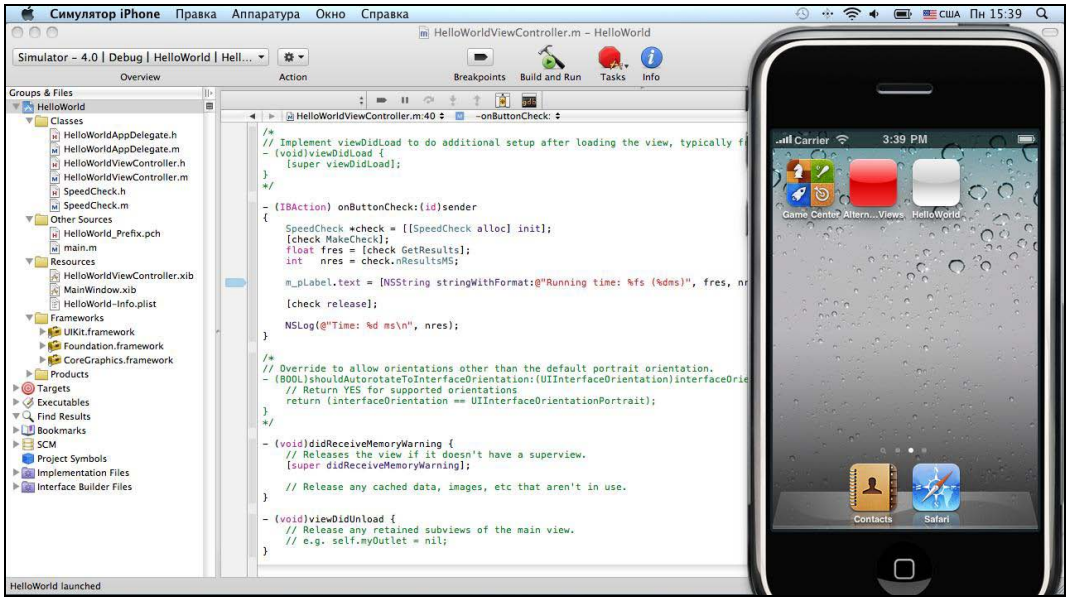


Рис. 2.5. Симулятор iPhone в среде XCode

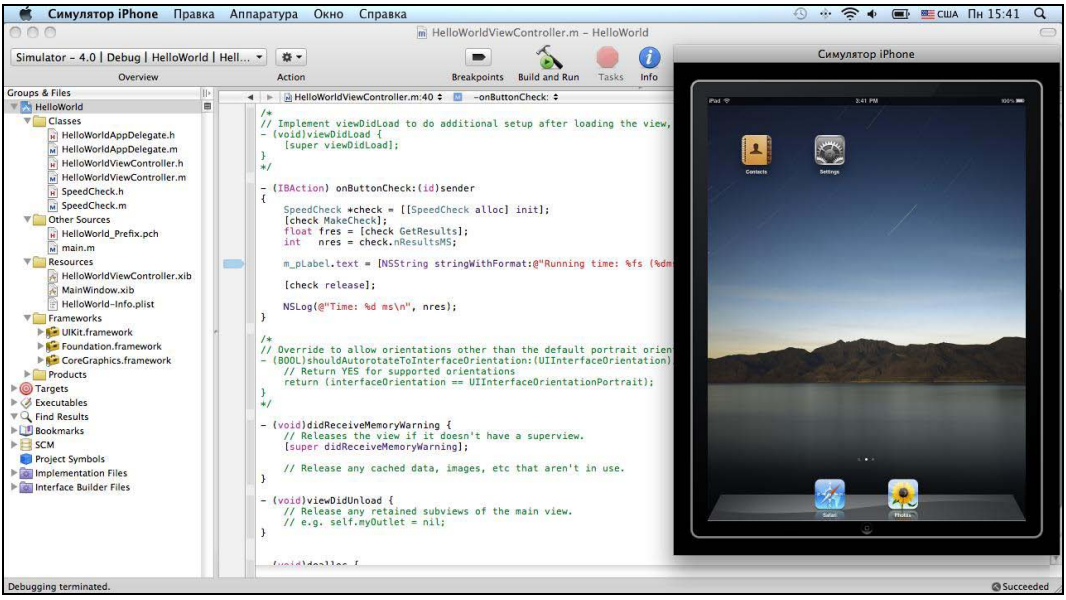


Рис. 2.6. Симулятор iPad в среде XCode

Симулятор iPad выглядит примерно так же (рис. 2.6), но его размеры соответственно больше, поэтому по умолчанию включается режим масштабирования до 50%.

Из-за большого размера диалоговых окон и ресурсов желательно иметь компьютер с монитором достаточного разрешения, не менее 1600×1200 пикселов.

Конечно, работу программ акселерометра трудно отлаживать с помощью симулятора, средств имитации наклона устройства он, к сожалению, не имеет. Существуют и еще некоторые ограничения, например отладка доступа к сервисам Apple, связанным с покупкой ПО (in app purchase). Тем не менее симулятор значительно облегчает разработку программы, и для большинства задач его функций более чем достаточно. Разобравшись со средой XCode, приступим к созданию первой программы для iOS.

Создаем программу Hello World

По традиции в качестве первой программы обычно выступает Hello World, и мы не будем эту традицию нарушать. Итак, запускаем среду разработки XCode, открываем окно создания нового проекта. Возможные варианты действий показаны на рис. 2.7.

Среда разработки XCode предлагает разработчику выбрать один из вариантов проекта:

- **Navigation-Based Application** — иерархическое представление открываемых окон;
- **OpenGL ES Application** — приложение для двумерной или трехмерной графики;
- **Tab Bar Application** — панель вкладок в качестве главного окна;

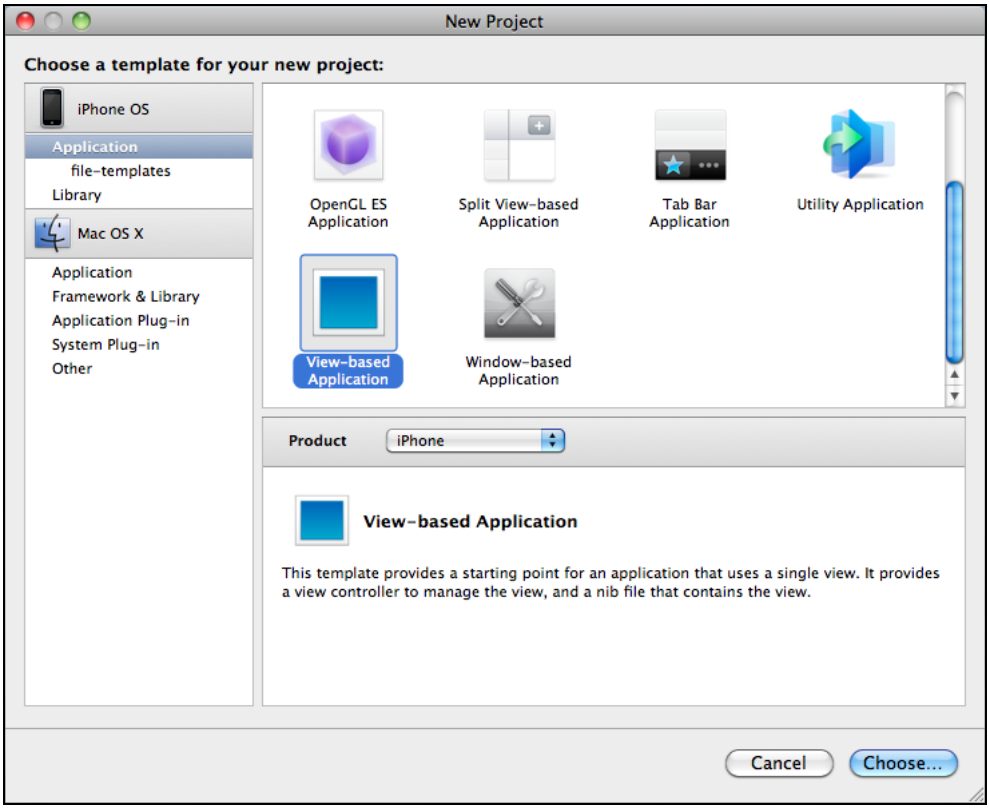


Рис. 2.7. Создание нового проекта

- **Utility Application** — приложение, состоящее из главного окна и окна настроек. Подходит в качестве начальной точки для изучения принципа переключения диалоговых окон в iOS;
- **View-based Application** — на базе класса `UIViewController`;
- **Window-based Application** — приложение, состоящее из главного окна без каких-либо классов.

Выберем тип **View-based Application** для iPhone. Затем необходимо задать местоположение файлов проекта и ввести название: `HelloWorld`.

Разберемся, для чего нужен каждый из создаваемых файлов.

Файлы HelloWorldAppDelegate

Эти файлы содержат класс `HelloWorldAppDelegate`, получающий сообщения от системного класса `UIApplicationDelegate`. Этот класс получает сообщения системы обо всех основных стадиях жизненного цикла программы — запуск, свертывание, развертывание и пр.

Обратимся к листингу 2.9 и рассмотрим эти файлы подробнее.

Листинг 2.9. Файл HelloWorldAppDelegate.h

```
@class HelloWorldViewController;
@interface HelloWorldAppDelegate : NSObject <UIApplicationDelegate>
{
    UIWindow *window;
    HelloWorldViewController *viewController;
}
@property (nonatomic, retain) IBOutlet UIWindow *window;
@property (nonatomic, retain) IBOutlet HelloWorldViewController
                                                *viewController;

@end
```

Как можно видеть, класс `HelloWorldAppDelegate` содержит два элемента: `window`, представляющий главное окно программы, и `viewController` — окно, видимое пользователю. С элементом `window` напрямую работать не требуется, он создается системой, а элемент `viewController` как раз и будет содержать все требуемые нам элементы интерфейса. Строка `<UIApplicationDelegate>` указывает, что класс рассчитан на получение сообщений системного интерфейса, подробное описание которых можно найти на сайте **http://developer.apple.com**. Рассмотрим те из них, которые были созданы для нового проекта (листинг 2.10).

Листинг 2.10. Файл HelloWorldAppDelegate.m

```
#import "HelloWorldAppDelegate.h"
#import "HelloWorldViewController.h"

@implementation HelloWorldAppDelegate

@synthesize window;
@synthesize viewController;

#pragma mark -
#pragma mark Application lifecycle

- (BOOL)application:(UIApplication *)application
            didFinishLaunchingWithOptions:(NSDictionary *)launchOptions
{
    // Override point for customization after application launch.
    // Add the view controller's view to the window and display.
    [window addSubview:viewController.view];
    [window makeKeyAndVisible];

    return YES;
```

```
}

- (void)applicationWillResignActive:(UIApplication *)application
{
 /*
     Sent when the application is about to move from active to inactive state. This can occur for certain types of temporary interruptions (such as an incoming phone call or SMS message) or when the user quits the application and it begins the transition to the background state.
     Use this method to pause ongoing tasks, disable timers, and throttle down OpenGL ES frame rates. Games should use this method to pause the game.
     */
}

- (void)applicationDidEnterBackground:(UIApplication *)application
{
    /*
     Use this method to release shared resources, save user data, invalidate timers, and store enough application state information to restore your application to its current state in case it is terminated later.
     If your application supports background execution, called instead of applicationWillTerminate: when the user quits.
     */
}

- (void)applicationWillEnterForeground:(UIApplication *)application
{
    /*
     Called as part of  transition from the background to the inactive state: here you can undo many of the changes made on entering the background.
     */
}

- (void)applicationDidBecomeActive:(UIApplication *)application
{
    /*
     Restart any tasks that were paused (or not yet started) while the application was inactive. If the application was previously in the background, optionally refresh the user interface.
     */
}

- (void)applicationWillTerminate:(UIApplication *)application
{
    /*
     Called when the application is about to terminate.
     See also applicationDidEnterBackground:.
```

```
     */
}

#pragma mark -
#pragma mark Memory management

- (void)applicationDidReceiveMemoryWarning:(UIApplication *)application
{
    /*
     Free up as much memory as possible by purging cached data objects that
can be recreated (or reloaded from disk) later.
     */
}

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

@end
```

Директива `import` вряд ли нуждается в описании, директивы `synthesize` нам уже знакомы — они служат для доступа к объектам `viewController` и `window`. Стоит также обратить внимание на удаление этих объектов внутри функции `dealloc`.

Директивы `#pragma mark` содержат комментарии к функциям, они являются опциональными.

Функция `application:didFinishLaunchingWithOptions` вызывается во время запуска программы. Ее можно и нужно использовать для инициализации дополнительных объектов. К окну программы с помощью функции `addSubview` добавляется `viewController` основного окна графического интерфейса программы. Здесь можно провести аналогию с функцией `InitInstance` класса `CWinApp` в MFC.

Функция `applicationDidEnterBackground` вызывается в тех случаях, когда программа свертывается в фоновый режим — пользователь нажимает кнопку **Home**, чтобы перейти к другой программе. Хорошим тоном при этом будет сохранение данных, освобождение ненужных ресурсов, приостановка таймеров и пр.

Функция `applicationDidBecomeActive`, напротив, вызывается во время активации программы, когда пользователь щелкнул ее значок.

Функция `applicationWillTerminate` вызывается по завершении программы. В iOS версии ниже 4.0 программа закрывается при нажатии клавиши **Home**, в 4.0 же программа только свертывается, а закрыть ее можно, войдя в специальный режим двойным нажатием кнопки **Home**.

Вернемся к функции `application:didFinishLaunchingWithOptions`, точнее, двум строкам внутри нее:

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

Именно здесь к приложению добавляется основное окно программы. Рассмотрим его класс более подробно.

Файлы HelloWorldViewController

В отличие от класса `AppDelegate`, который создается один раз и больше практически не меняется, класс `HelloWorldViewController` собственно и представляет функциональность главного окна программы (листинг 2.11).

Листинг 2.11. Файл HelloWorldViewController.h

```
//  HelloWorldViewController.h
//

#import <UIKit/UIKit.h>

@interface HelloWorldViewController : UIViewController
{

}

@end
```

Как можно видеть, заголовочный файл пока пуст, он не содержит ничего, кроме описания класса, наследуемого от `UIViewController`. Но это ненадолго, при добавлении в окно дополнительных компонентов они будут вписываться именно сюда (листинг 2.12).

Листинг 2.12. Файл HelloWorldViewController.m

```
//  HelloWorldViewController.m
//

#import "HelloWorldViewController.h"

@implementation HelloWorldViewController

/*
// The designated initializer. Override to perform setup that is required
// before the view is loaded.
- (id)initWithNibName:(NSString *)nibNameOrNil
               bundle:(NSBundle *)nibBundleOrNil
```

```
{
    if ((self = [super initWithNibName:nibNameOrNil
                        bundle:nibBundleOrNil])) {
        // Custom initialization
    }
    return self;
}
*/

/*
// Implement loadView to create a view hierarchy programmatically,
// without using a nib.
- (void)loadView
{
}
*/

/*
// Implement viewDidLoad to do additional setup after loading the view, typi-
cally from a nib.
- (void)viewDidLoad
{
    [super viewDidLoad];
}
*/

/*
// Override to allow orientations other than the default portrait orientation.
-
(BOOL)shouldAutorotateToInterfaceOrientation:(UIInterfaceOrientation)interface
Orientation
{
    // Return YES for supported orientations
    return (interfaceOrientation == UIInterfaceOrientationPortrait);
}
*/

- (void)didReceiveMemoryWarning
{
    // Releases the view if it doesn't have a superview.
    [super didReceiveMemoryWarning];

    // Release any cached data, images, etc that aren't in use.
}

- (void)viewDidUnload
```

```
{
    // Release any retained subviews of the main view.
    // e.g. self.myOutlet = nil;
}


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

@end
```

Несколько полезных функций, которые обязательно пригодятся нам в дальнейшем, рассмотрим подробнее:

- `viewDidLoad` — вызывается после загрузки окна в память (вроде `OnInitDialog` в MFC). Если необходима дополнительная инициализация элементов управления (например, загрузка данных из предыдущего сеанса работы), ее удобно разместить здесь;
- `viewDidUnload` — вызывается в конце жизненного цикла окна;
- `shouldAutorotateToInterfaceOrientation` — вызывается системой при повороте устройства, разработчик должен вернуть `TRUE`, если приложение поддерживает разворот экрана в этом направлении, `FALSE` — если нет. Если функция вернула `FALSE`, то при перевороте смартфона приложение останется "как есть", а если функция вернула `TRUE`, окно программы тоже развернется, но элементы управления могут при этом сдвинуться. Всего доступно 4 варианта положения окон интерфейса:

  ```
  typedef enum {
     UIInterfaceOrientationPortrait           = UIDeviceOrientationPortrait,
     UIInterfaceOrientationPortraitUpsideDown = UIDeviceOrientationPortrai-
  tUpsideDown,
     UIInterfaceOrientationLandscapeLeft      = UIDeviceOrientationLandscape-
  Right,
     UIInterfaceOrientationLandscapeRight     = UIDeviceOrientationLandscape-
  Left
  } UIInterfaceOrientation;
  ```

  Пользователь может выбирать, какой вид ориентации поддерживать в программе, возвращая `TRUE` для нужных значений. Более подробно о развороте экрана будет сказано *в главе 6*;
- `didReceiveMemoryWarning` — вызывается в том случае, если в системе не хватает памяти;
- `dealloc` — все используемые компоненты (кнопки, текстовые поля) должны быть освобождены в этой функции.

В дальнейшем мы еще вернемся к этому файлу, когда будем наращивать функциональность приложения.

Файл HelloWorld_Prefix.pch

Расширение этого файла можно рассматривать как шутку разработчиков XCode, поскольку в Visual Studio создаются временные файлы с таким же расширением, и их обычно удаляют. В XCode, наоборот, этот файл важен в проекте, что можно видеть в листинге 2.13.

Листинг 2.13. Файл HelloWorld_Prefix.pch

```
//
// Prefix header for all source files of the 'HelloWorld' target in the
'HelloWorld' project
//

#ifdef __OBJC__
    #import <Foundation/Foundation.h>
    #import <UIKit/UIKit.h>
#endif
```

Файл содержит заголовки, включаемые во все файлы проекта. Изменять этот файл приходится довольно редко, но очевидно, что без него программа компилироваться не будет. На что и натолкнулся автор в самом же начале изучения программирования под iOS, когда освободил папку от "лишних" файлов, включая и pch.

Файл main.m

Этот файл содержит функцию `main`, вид которой практически совпадает с описанием функции `main` в языке C (листинг 2.14).

Листинг 2.14. Файл main.m

```
//  main.m
//  HelloWorld
//
//  Created on 10.12.10.
//  Copyright 2010. All rights reserved.
//

#import <UIKit/UIKit.h>

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

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

Этот файл обычно не приходится изменять, тем не менее, интересно разобраться, что в нем делается. Класс `NSAutoreleasePool` используется для обеспечения управления памятью, его описание можно найти на сайте **http://developer.apple.com**.

Самой функции `UIApplicationMain` в проекте нет, видимо она находится в системных файлах SDK. Еще одна загадка состоит в том, что объект `pool` создается, но далее не используется. Впрочем, вполне вероятно, что внутри `NSAutoreleasePool` находится какой-либо статический указатель, и системные классы обращаются к нему через статическую функцию.

Файл HelloWorldViewController.xib

Этот XML-файл содержит описание окна и всех его компонентов. В листинге 2.15 мы приведем его начало.

Листинг 2.15. Файл HelloWorldViewController.xib

```
<?xml version="1.0" encoding="UTF-8"?>
<archive type="com.apple.InterfaceBuilder3.CocoaTouch.XIB" version="7.10">
        <data>
                <int key="IBDocument.SystemTarget">1024</int>
                <string key="IBDocument.SystemVersion">10D2125</string>
                <string  key="IBDocument.InterfaceBuilderVersion"> 788
                </string>
                <string key="IBDocument.AppKitVersion">1038.29</string>
                <string key="IBDocument.HIToolboxVersion">460.00</string>
                <object class="NSMutableDictionary" key =
                                        "IBDocument.PluginVersions">
                        <string key = "NS.key.0">
                            com.apple.InterfaceBuilder.IBCocoaTouchPlugin
                        </string>
                        <string key="NS.object.0">117</string>
                </object>
…
</archive>
```

Для редактирования ресурсов пользователь должен запомнить все типы полей и константы… Нет, конечно же, это шутка, вряд ли разработчику придется изменять этот файл вручную, вся обработка ресурсов успешно делается с помощью Interface Builder.

К HelloWorldViewController.xib мы вернемся позже, когда будем добавлять элементы управления в программу.

Файл MainWindow.xib

Этот файл содержит главное окно программы, задача которого состоит лишь в том, чтобы быть "оберткой" для окна `HelloWorldViewController`. Никаких элементов управления непосредственно в MainWindow.xib в этом проекте добавлять не придется.

Файл HelloWorld-Info.plist

Этот файл важен для проекта настолько, что однозначно заслуживает не только отдельного раздела, а, может, и отдельной главы. Небольшой XML-файл содержит описание всех основных параметров программы, использующихся как операционной системой для запуска, так и сервисом Apple Appstore для загрузки программы.

Содержимое файла HelloWorld-Info.plist можно видеть в листинге 2.16.

Листинг 2.16. Файл HelloWorld-Info.plist

```
<?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>English</string>
        <key>CFBundleDisplayName</key>
        <string>${PRODUCT_NAME}</string>
        <key>CFBundleExecutable</key>
        <string>${EXECUTABLE_NAME}</string>
        <key>CFBundleIconFile</key>
        <string></string>
        <key>CFBundleIdentifier</key>
        <string>com.yourcompany.${PRODUCT_NAME:rfc1034identifier}</string>
        <key>CFBundleInfoDictionaryVersion</key>
        <string>6.0</string>
        <key>CFBundleName</key>
        <string>${PRODUCT_NAME}</string>
        <key>CFBundlePackageType</key>
        <string>APPL</string>
        <key>CFBundleSignature</key>
        <string>????</string>
        <key>CFBundleVersion</key>
        <string>1.0</string>
        <key>LSRequiresIPhoneOS</key>
        <true/>
        <key>NSMainNibFile</key>
        <string>MainWindow</string>
</dict>
</plist>
```

К счастью, как и в случае с редактором ресурсов, изменять этот файл вручную не требуется, в состав XCode входит встроенный редактор plist-файла, его внешний вид показан на рис. 2.8.

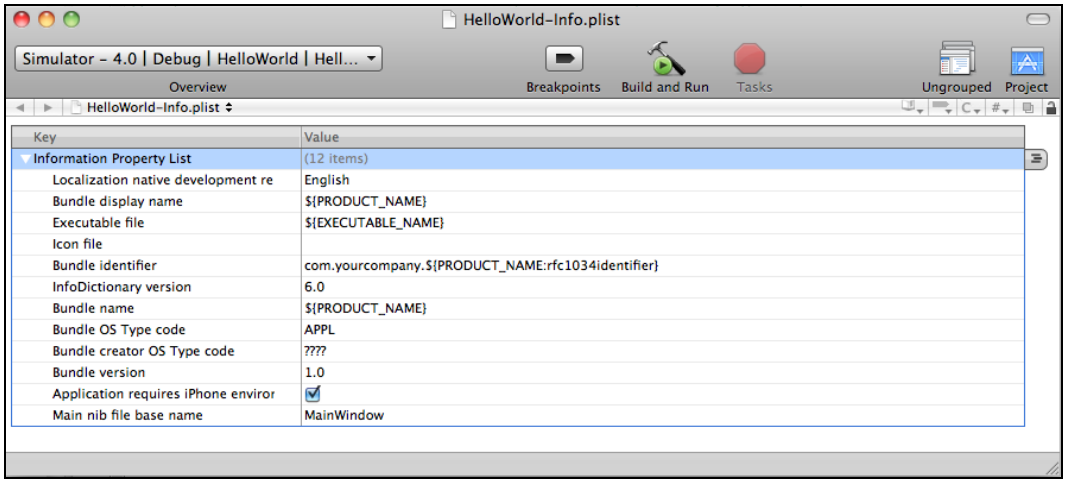


Рис. 2.8. Редактор plist-файлов

Рассмотрим наиболее важные поля подробнее:

- `CFBundleDisplayName` — имя программы, которое будет отображаться на экране iPhone или iPad;
- `CFBundleIconFile` — имя файла со значком программы на экране iPhone. Значок хранится в png-файле и имеет размер 57×57 пикселов;
- `CFBundleIconFiles` — список файлов значков для iPhone 4 и iPad. Значок для iPhone 4 должен иметь размеры 114×114 пикселов, для iPad — 72×72. Оба эти параметра (`CFBundleIconFile` и `CFBundleIconFiles`) важны для размещения программы в App Store;
- `CFBundleIdentifier` — уникальный идентификатор приложения, обычно состоящий из трех частей, например `app.book.helloworld`. Рекомендуется задавать это имя в виде имени сайта и приложения в обратном порядке, например, `com.mysite.myappname`, в таком случае будет меньше путаницы в названиях при большом количестве программ. Важно помнить, что после отправки в App Store поменять идентификатор программы уже невозможно, к нему привязаны все цифровые сертификаты, так что к выбору `CFBundleIdentifier` стоит отнестись внимательно;
- `CFBundleVersion` — номер версии, который также важен для загрузки в App Store. При выпуске обновлений в App Store номер версии необходимо увеличить в этом поле, в противном случае программа не будет принята;
- `NSMainNibFile` — имя ресурсов главного окна приложения. Изменять этот параметр вряд ли придется;

- `UISupportedInterfaceOrientations` — опциональный, показывает, какие виды положения экрана поддерживает программа. Если программа рассчитана для работы только в определенной ориентации экрана (например, горизонтальной), необходимо прописать возможные варианты в полях этого ключа;
- `UIStatusBarHidden` — опциональный параметр, указывает на необходимость скрытия панели статуса. Как уже упоминалось ранее, панель статуса является важным элементом системы, и скрывать ее целесообразно только в специальных случаях, например для игровых программ;
- `UIStatusBarStyle` — позволяет задать стиль панели статуса, например при необходимости его можно сделать белым или черным, в зависимости от стиля разрабатываемого приложения. Доступны три варианта значений: `UIStatusBarStyleDefault` (по умолчанию), `UIStatusBarStyleBlackTranslucent` (полупрозрачный с черным) и `UIStatusBarStyleBlackOpaque` (непрозрачный с черным);
- `UIRequiredDeviceCapabilities` — возможности устройства, задействованные программой. Например, программа может использовать камеру, GPS-приемник или компас, которые могут присутствовать в одних устройствах и отсутствовать в других.

Полный список констант plist-файлов велик, подробное описание можно найти в разделе "About Information Property List Files" сайта **http://developer.apple.com**.

После того как мы разобрались со всеми файлами проекта, самое время скомпилировать его и посмотреть, что получилось. Выбираем режим **Simulator** и конфигурацию **Debug** — щелкаем кнопку **Build and Run**. То, что получилось, можно видеть на рис. 2.9.

Рис. 2.9. Первая программа для iPhone

Можно с удовлетворением взглянуть на плоды своих трудов, однако это еще не все. Первый вопрос, который может возникнуть — как загрузить программу не на симулятор, а на настоящий iPhone. Как уже говорилось, это непросто. Для того чтобы иметь возможность загрузки программы на устройство, нужен цифровой сертификат, загрузить который можно, лишь получив лицензию разработчика Apple, которая стоит 100 долларов в год. Впрочем, в сети Интернет описывались способы бесплатной генерации сертификата, интересующиеся могут заняться этой темой самостоятельно. Конечно, "самодельный" сертификат не может быть использован для загрузки программ в App Store, но для тестирования на iPhone или iPad такой метод вполне может пригодиться. Тем более что российское законодательство официально разрешает адаптацию ПО, под которой понимается "внесение изменений, осуществляемых исключительно в целях обеспечения функционирования программы для ЭВМ на конкретных технических средствах пользователя". И естественно, даже с точки зрения здравого смысла, глупо запрещать разработчику запускать его собственную программу на его собственном смартфоне. Но даже если удастся запустить программу без сертификата, ее отладка все равно будет невозможна. Так что для серьезной разработки программ сертификат от Apple все-таки придется приобрести, подробнее об этом будет рассказано *в главе 10*.

Полюбовавшись на нашу программу Hello World, заметим, что она, как это ни странно, ничего полезного не делает. Настало время перейти к наполнению ее полезным содержанием, для чего придется разобраться с редактированием ресурсов программы и их привязкой к коду.

Создание и редактирование интерфейса программы

Для начала найдем в проекте файл HelloWorldViewController.xib и дважды щелкнем его. Откроется окно редактора Interface Builder с четырьмя панелями (рис. 2.10):

- **HelloWorldViewController.xib**;
- **View** — просмотр экрана смартфона;
- **Library** — библиотека компонентов;
- **View Attributes** — свойства выбранного элемента, в нашем случае окна.

Наибольший интерес для нас будут представлять сейчас левое окно **Library**, показывающее список доступных элементов интерфейса, и правое окно, показывающее свойства компонентов.

Выберем в библиотеке компонент **Label** и разместим его в окне **View**. Если дважды щелкнуть значок **Label**, появится окно редактирования, в котором можно ввести текст.

Введем текст `Hello world`, нажмем клавишу <Enter> и сохраним результаты нажатием комбинации клавиш <Cmd>+<S>. Теперь можно переключиться в XCode и снова скомпилировать проект (рис. 2.11).

Функциональность программы возросла на порядок, и теперь это уже не "пустое" приложение. Но пока еще программа ничего не делает, нужно добавить интерактивные элементы управления. Начнем с кнопки.

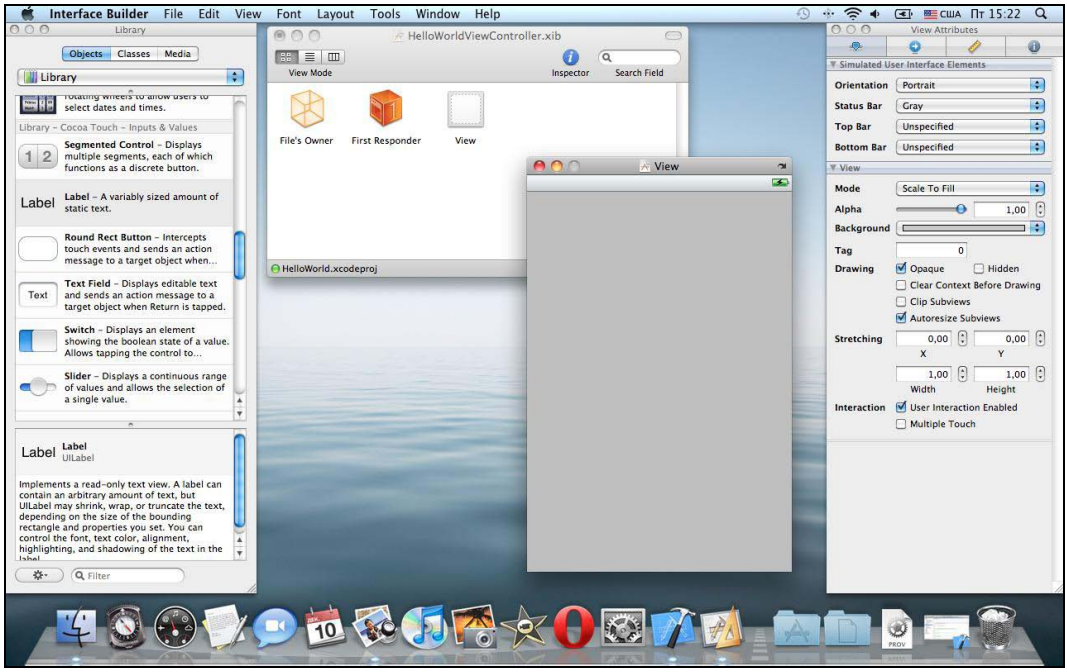


Рис. 2.10. Внешний вид Interface Builder

Рис. 2.11. Программа Hello World в симуляторе

Переключившись обратно в Interface Builder, выберем в окне **Library** элемент **Round Rect Button**. Далее необходимо сделать, чтобы эта кнопка выполняла какую-либо функцию. Ранее, в листинге 2.8, мы создали класс, позволяющий измерять время выполнения программы, этим кодом мы и воспользуемся. Для начала поместим кнопку в окно **View**, поменяем ее текст на `Check speed`. Однако это еще не все. Для того чтобы связать кнопку с кодом программы, нужно добавить функцию обработки нажатия кнопки в класс `HelloWorldViewController` и связать функцию-обработчик с графическим интерфейсом.

Переключимся в XCode и добавим в класс `HelloWorldViewController` строку описания функции, как показано в листинге 2.17.

Листинг 2.17. Класс `HelloWorldViewController`

```
@interface HelloWorldViewController : UIViewController
{

}

- (IBAction) onButtonTest:(id)sender;

@end
```

Жирным шрифтом выделена строка, которую необходимо добавить. Макрос `IBAction` не случайно начинается с букв `IB`, по нему Interface Builder определяет наличие обработчика для связывания в дальнейшем.

Теперь добавим тело функции в файл HelloWorldViewController.m. После добавления вставим в нее код нашего класса, который мы писали ранее (листинг 2.18).

Листинг 2.18. Класс `HelloWorldViewController`

```
- (IBAction) onButtonTest:(id)sender
{
        SpeedCheck *check = [[SpeedCheck alloc] init];
        [check MakeCheck];
        float fres = [check GetResults];
        int   nres = check.nResultsMS;

        [check release];
}
```

Для того чтобы код скомпилировался без ошибок, в начало файла надо не забыть вставить строку `#import "SpeedCheck.h"`. Файлы SpeedCheck.h и SpeedCheck.m должны быть включены в проект с помощью команды **Add file**. Теперь можно поставить точку останова внутри функции, запустить и посмотреть результат.

Запускаем симулятор, нажимаем кнопку — и ничего не происходит. Неудивительно, т. к. необходимо установить связь между окном **View** и нашей функцией-обработчиком.

Итак, снова переключаемся в Interface Builder, не забыв сохранить изменения в коде нажатием комбинации клавиш <Cmd>+<S>. Если не сохранить файлы, то Interface Builder не "увидит" обновленные файлы, и установить связь не получится.

Для того чтобы установить связь, нужно сделать следующее.

1. Открыть Interface Builder, выбрать созданную на предыдущем этапе кнопку и правым щелчком на ней открыть контекстное меню.
2. Выбрать в контекстном меню обработчик (**Touch Up Inside**) и сделать самое интересное — щелкнуть мышью кружок против надписи **Touch Up Inside**, затем провести стрелку от него до значка **File's Owner**. Как это должно выглядеть, показано на рис. 2.12. Если функция обработчика нажатия кнопки существует в данном классе, ее имя должно появиться на экране.

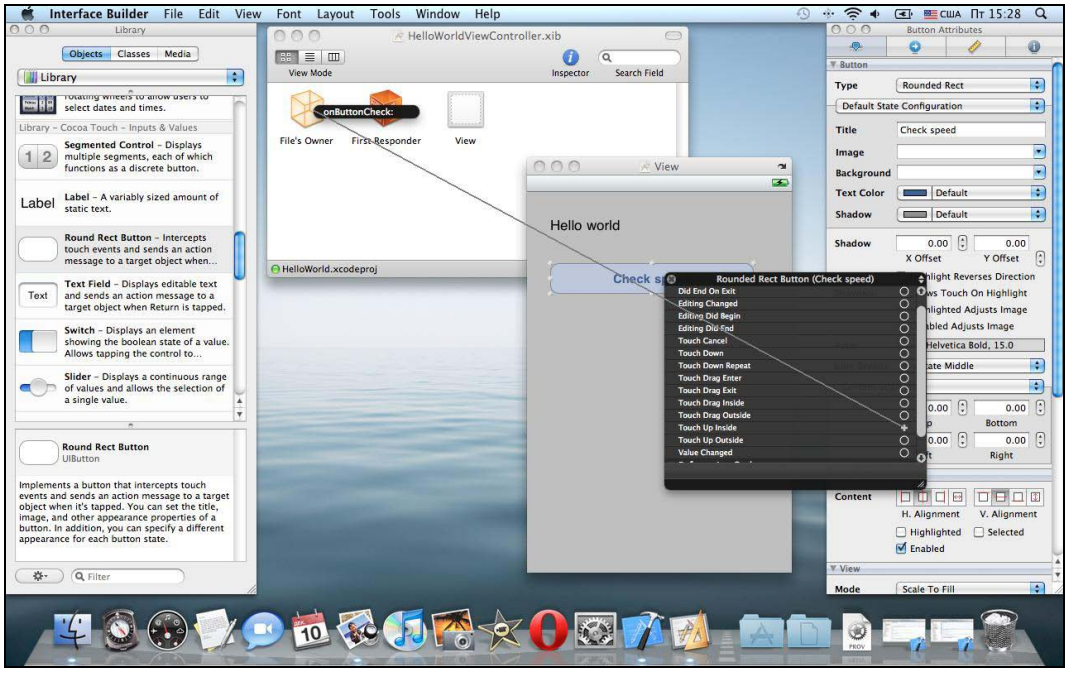


Рис. 2.12. Связывание кнопки и обработчика в Interface Builder

Теперь контекстное меню должно выглядеть, как на рис. 2.13.

Если обработчик не появляется, значит, один из шагов пропущен: не создано описание функции в h-файле, не дописан тип `IBOutlet`, файл не сохранен на диске либо предпринимается попытка связать кнопку с другим файлом **View**.

Когда все сделано, можно сохранить изменения в Interface Builder, переключиться в XCode и еще раз запустить программу, она должна корректно выполняться.

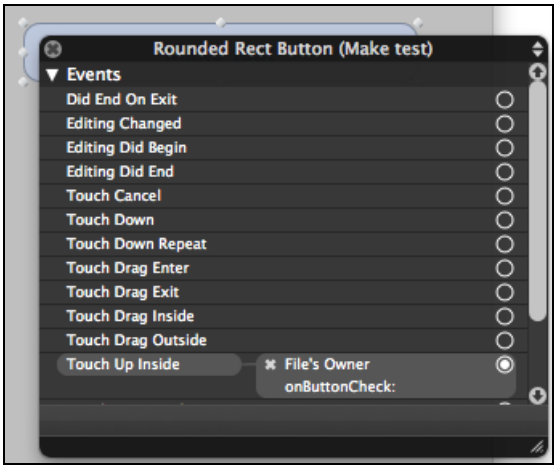


Рис. 2.13. Контекстное меню кнопки

Однако от программы, которая ничего не выводит, толку не так много. Теперь нужно научиться делать обратную операцию — выводить данные на экран. Привязывать функции к обработчику мы уже умеем, теперь нужно аналогичным методом сделать привязку элементов интерфейса к членам класса `HelloWorldViewController`.

Конечно, можно было бы создать отдельное поле вывода данных, но для упрощения задачи можно вывести данные прямо в поле метки, которое изначально подписано **Hello world**. Для этого нужно выполнить несколько действий.

1. Добавить переменную типа `UILabel` в класс `HelloWorldViewController`.
2. Добавить свойство (`property`), чтобы программа могла иметь автоматический доступ к созданной переменной.
3. Добавить поля для синтеза переменной и ее удаления.
4. Не забыть сохранить файлы проекта и, переключившись в Interface Builder, установить связь между созданным членом класса и элементом окна.
5. Использовать переменную по нашему усмотрению.

Рассмотрим эти шаги подробнее. Нам надо добавить в класс переменную, описав ее как свойство (`property`).

Листинг 2.19. Класс `HelloWorldViewController`

```
@interface HelloWorldViewController : UIViewController
{
        IBOutlet UILabel        *m_pLabel;
}

@property(retain, nonatomic) UILabel        *m_pLabel;

- (IBAction) onButtonCheck:(id)sender;

@end
```

В листинге 2.19 выделены строки, которые нужно добавить. Мы использовали дополнительные параметры директивы `@property`, их описание можно найти в документации по Objective-C.

Далее необходимо добавить соответствующий код в m-файл, как показано в листинге 2.20.

Листинг 2.20. Файл HelloWorldViewController.m

```
#import "HelloWorldViewController.h"
#import "SpeedCheck.h"

@implementation HelloWorldViewController
@synthesize m_pLabel;
…
- (void)dealloc
{
    [m_pLabel release];

    [super dealloc];
}

@end
```

Здесь была добавлена директива `@synthesize` для создания класса, а в методе `dealloc` был добавлен вызов `release` для корректного освобождения памяти.

Следующий шаг — привязка созданного члена класса к интерфейсу (рис. 2.14).

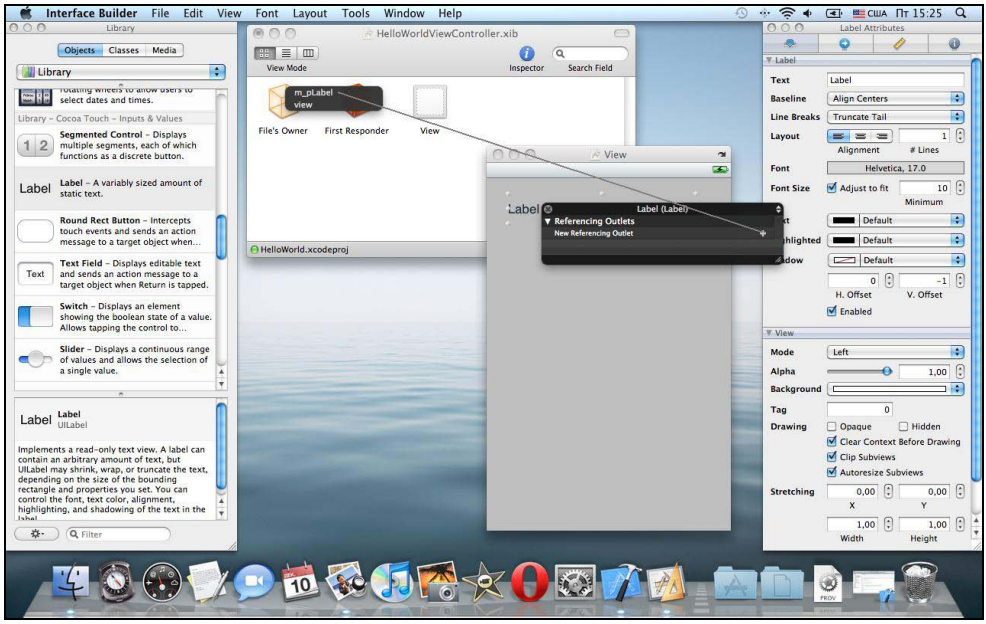


Рис. 2.14. Привязка `m_pLabel` к интерфейсу

Как и в предыдущем случае, необходимо выбрать в контекстном меню элемента **Label** пункт **New Referencing Outlet** и протянуть мышью линию от плюса к элементу **File's Owner**. Если при наведении мыши на пункт **File's Onwer** метка **m_pLabel** не появляется, значит, что-либо на предыдущем шаге было сделано неверно.

Если все было сделано правильно, контекстное меню метки **Label** должно выглядеть, как на рис. 2.15.

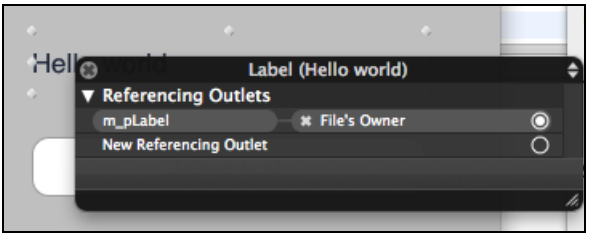


Рис. 2.15. Контекстное меню элемента **Label**

Теперь можно сохранить изменения и переключиться обратно в Xcode. Переменная `m_pLabel` привязана к интерфейсу, и мы можем использовать ее в коде программы.

Окончательный вариант функции-обработчика показа в листинге 2.21.

Листинг 2.21. Обработчик нажатия кнопки

```
- (IBAction) onButtonCheck:(id)sender
{
      SpeedCheck *check = [[SpeedCheck alloc] init];
      [check MakeCheck];
      float fres = [check GetResults];
      int   nres = check.nResultsMS;

      m_pLabel.text = [NSString stringWithFormat:
                           @"Running time: %fs (%dms)", fres, nres];

      [check release];
}
```

Мы воспользовались свойством `text` элемента `m_pLabel`, и благодаря установленной связи с интерфейсом изменение этого поля приводит к изменению на экране. Результат показан на рис. 2.16. Теперь эта программа — уже вполне полезный измерительный инструмент, с помощью которого можно оценить быстродействие выполняемого кода в разных режимах.

Результаты показаны в табл. 2.1.

Таблица 2.1. *Время выполнения функции* `onButtonCheck`

Конфигурация проекта	**Время выполнения, мс**
Simulator, Debug-версия	1050
Simulator, Release-версия	1007
iPad, Debug-версия	5328
iPad, Release-версия	4913

Симулятор работал на ноутбуке MacBook Pro, оснащенном процессором Core 2 Duo P8600 с тактовой частотой 2,4 ГГц (Apple iPad, как известно, имеет частоту процессора 1 ГГц). Интересно заметить, что по скорости вычислений Core 2 Duo практически в 5 раз превосходит процессор ARM Cortex A4, установленный в iPad. С другой стороны, это очень даже хороший результат, учитывая, что процессор iPad работает на меньшей тактовой частоте и потребляет всего лишь 2,5 Вт мощности, не требуя при этом никакого активного охлаждения.

Таким образом, мы успешно создали и запустили программу для iPhone. Однако если мы запустим ее на iPad, то увидим картинку, показанную на рис. 2.17.

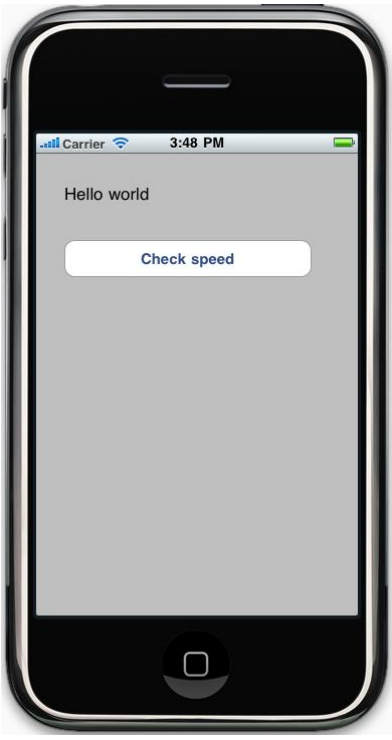


Рис. 2.16. Окончательный вариант программы HelloWorld

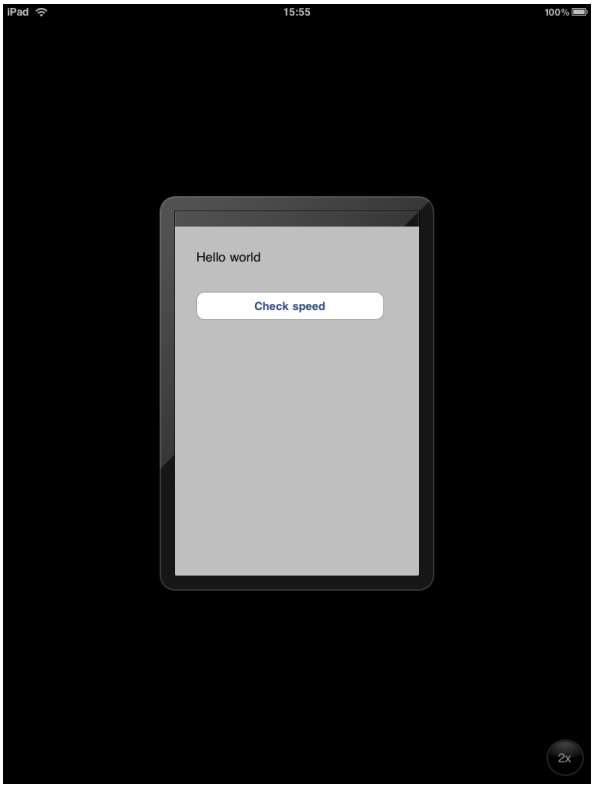


Рис. 2.17. Программа HelloWorld на iPad

iPhone-программа запускается на iPad в режиме эмуляции смартфона, что на большом экране выглядит некрасиво. Для того чтобы исправить положение, нужно указать в настройках проекта, что наша программа может работать и на iPhone, и на iPad. Для этого в свойствах проекта нужно найти пункт **Targeted Device Family** и установить в нем значение **iPhone/iPad**.

После перекомпиляции и загрузки на устройство вид программы похож на рис. 2.18.

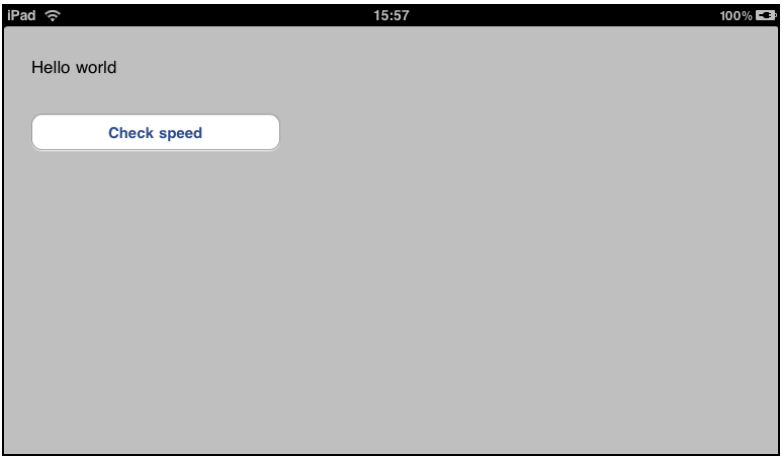


Рис. 2.18. Окончательный вариант программы HelloWorld с поддержкой iPhone и iPad

Теперь наша работа полностью завершена, программа корректно функционирует на устройствах под управлением iOS. Можно отдохнуть и перейти к изучению методов отладки в среде XCode.

Отладка приложений в XCode

При разработке программ возможны два способа отладки. Первый — это использование уже упоминавшейся функции `NSLog`, вывод информации происходит в отдельную консоль.

Добавим в функцию `onButtonCheck` строку вывода `NSLog`, как показано в листинге 2.22.

Листинг 2.22. Использование `NSLog`

```
- (IBAction) onButtonCheck:(id)sender
{
      SpeedCheck *check = [[SpeedCheck alloc] init];
      [check MakeCheck];

      float fres = [check GetResults];
```

```
    int   nres = check.nResultsMS;

    m_pLabel.text = [NSString stringWithFormat:
                     @"Running time: %fs (%dms)", fres, nres];

    [check release];

    NSLog(@"Time: %d ms\n", nres);
}
```

Запустим программу и откроем консоль рядом с симулятором. Результат показан на рис. 2.19, при нажатии на кнопку в окне **Debugger Console** будет появляться новая строка.

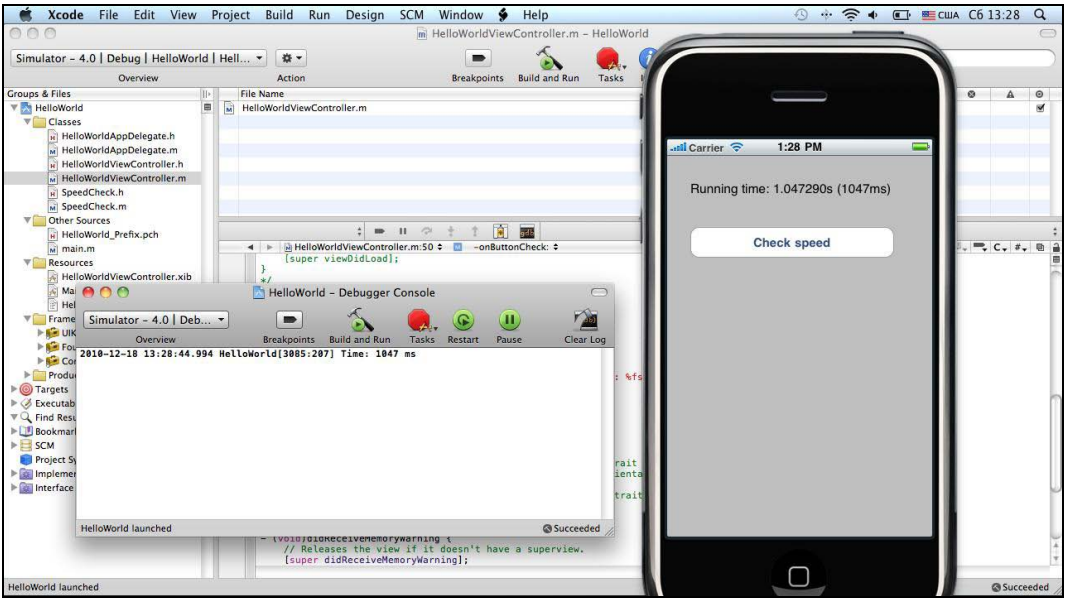


Рис. 2.19. Отладка с помощью `NSLog`

Такой способ отладки имеет два преимущества:

- не прерывается выполнение программы. Это может быть удобно при отладке различных интерактивных модулей, например, функций, взаимодействующих с экраном или акселерометром;
- данные, выведенные при помощи `NSLog`, остаются и при закрытии программы, что позволяет использовать их позже.

Конечно, использовать только такой способ отладки не очень удобно, но в XCode есть встроенный отладчик (рис. 2.20).

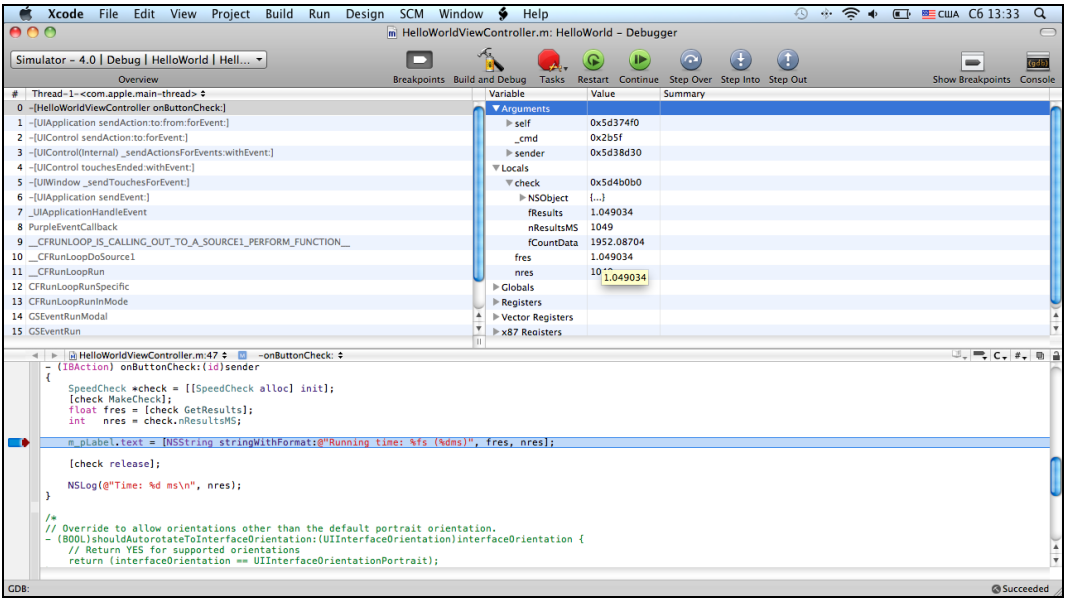


Рис. 2.20. Отладка с помощью XCode

Более подробно с функциями отладки можно познакомиться в документации к XCode, а пока что остановимся на еще одном важном аспекте работы с памятью.

Проверка утечек памяти с помощью XCode

Поскольку практически все операции с объектами в Objective-C требуют работы с памятью, актуальным является поиск и отслеживание утечек памяти. Для этого XCode предлагает целых два способа.

Первый — встроенный анализатор кода, активировать который можно с помощью пункта меню **Build and Analyze**. Для проверки возможностей программы закомментируем строку `release`, как показано в листинге 2.23.

Листинг 2.23. Функция с утечками памяти

```
- (IBAction) onButtonCheck:(id)sender
{
      SpeedCheck *check = [[SpeedCheck alloc] init];
      [check MakeCheck];
      float fres = [check GetResults];
      int   nres = check.nResultsMS;

      m_pLabel.text = [NSString stringWithFormat:
                        @"Running time: %fs (%dms)", fres, nres];
```

```
    // [check release]; // комментирование этой строки вызывает утечку

    NSLog(@"Time: %d ms\n", nres);
}
```

Результат показан на рис. 2.21. Еще до запуска приложения XCode точно показал строку, за которой объект уже не используется, как сказано в сообщении.

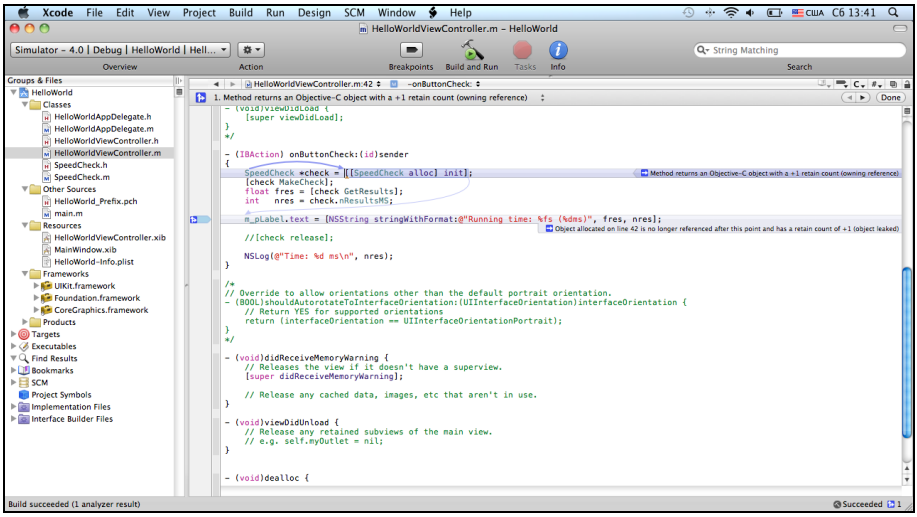


Рис. 2.21. Отображение потенциальных утечек памяти с помощью XCode

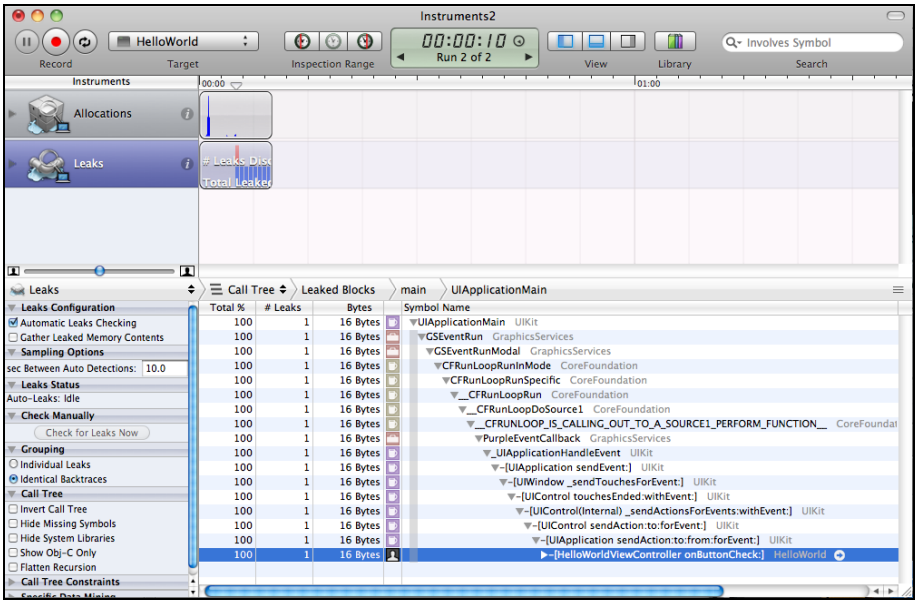


Рис. 2.22. Отображение утечек памяти с помощью программы Leaks

Конечно, такой способ анализа удобен, но, к сожалению, он срабатывает не всегда. Для более сложных случаев служит анализ выделения памяти во время выполнения программы — программа Leaks, вызвать которую можно из меню XCode.

Внешний вид окна Leaks показан на рис. 2.22. Утилита позволяет запустить тестируемую программу, в реальном времени показывает процесс выделения и освобождения памяти. Это может быть достаточно удобно, как для тестирования количества расходуемой памяти, так и для поиска утечек. По завершении работы можно посмотреть лог выполнения.

Можно просмотреть стек вызовов функций и достаточно точно локализовать место утечки.

На этом мы заканчиваем обзор основ программирования для iOS и будем постепенно переходить к более специализированным темам.

Глава 3

Взаимодействие программы и пользователя — основы

Редактор Interface Builder среды XCode

Кратко мы уже ознакомились с редактором Interface Builder в предыдущей главе, рассмотрим его возможности более подробно.

Настройка параметров компонентов

Для каждого из компонентов можно настроить разнообразные параметры с помощью модуля Inspector. Окно **Inspector** состоит из 4 панелей (рис. 3.1):

- **Label attributes** — основные параметры компонента (выравнивание текста, шрифт и размер текста, цвет фона, прозрачность);

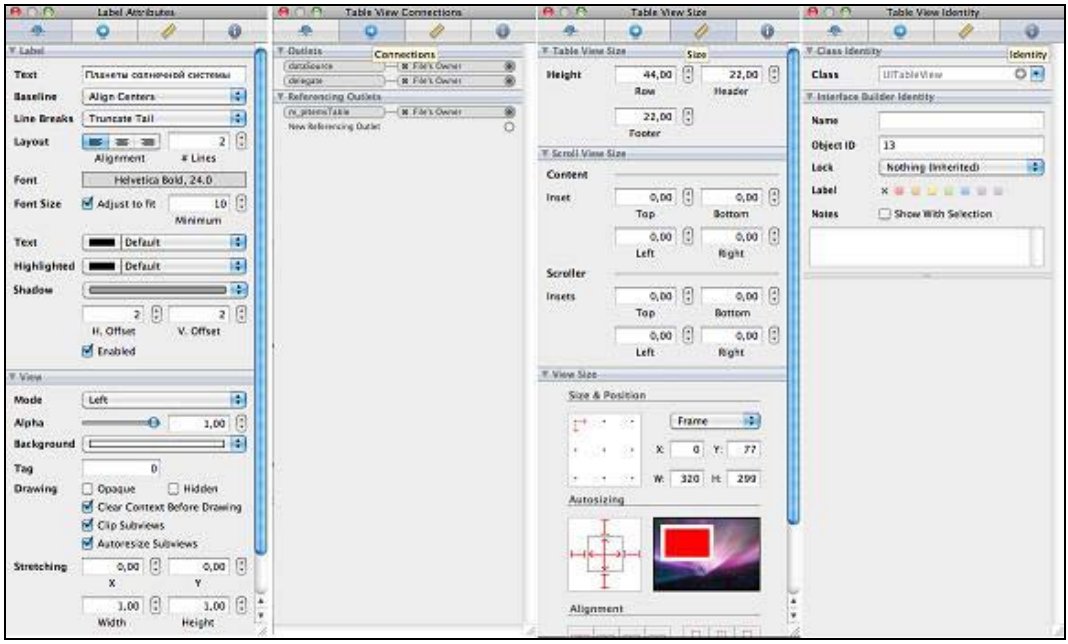


Рис. 3.1. Настройка параметров с помощью Interface Builder

- **Table View connections** — обеспечивает связывание данных кода и интерфейса, что осуществляется с помощью трех объектов:
  - `dataSource` — класс, являющийся источником данных для компонента со сложной структурой, например Table View;
  - `delegate` — класс, который будет получать сообщения от компонента, например, чтобы при вводе данных соответствующим образом менялись результаты, выводимые программой;
  - `referencing outlet` — член класса, связанный с выбранным компонентом;
- **Table View size** — размер и местоположение относительно границ экрана;
- **Table View Identity** — настройка классов для связывания объектов интерфейса и исходного кода. При использовании наследуемых классов (например, `UIView`) необходимо ввести в поле **Class** значение нового класса, в противном случае связь не будет установлена.

Для программы на рис. 3.2 были заданы размер и тип шрифта элементов `UILabel`, фоновый рисунок для окна с режимом растяжения на все окно. В **Table View** был выбран прозрачный цвет фона **Clear Color**, что позволило вывести текст поверх фонового изображения. Для всех элементов интерфейса были указаны режимы выравнивания, чтобы элементы корректно масштабировались при повороте экрана. Весь код в этом примере (кроме заполнения таблицы названиями планет) — добавление функции `shouldAutorotateToInterfaceOrientation`, разрешающей поворот экрана.

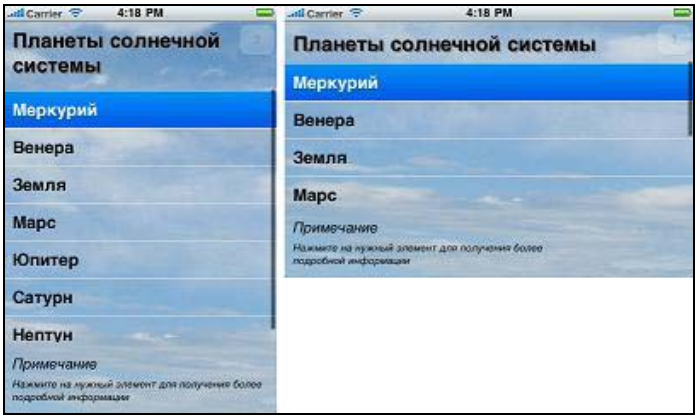


Рис. 3.2. Пример интерфейса, созданного с помощью Interface Builder

Краткий обзор пользовательского интерфейса iOS

Общая характеристика большинства компонентов была дана в предыдущей главе, теперь перейдем к деталям на практике.

Кнопка (*UIButton*)

Для обеспечения доступа к объекту `UIButton` следует объявить переменную:

```
IBOutlet UIButton  *m_cButton;
@property(nonatomic, retain) UIButton  *m_cButton;
```

Операционная система позволяет задать для кнопки следующие свойства:

- Название.

```
[m_cButton setTitle:@"setTitle" forState:UIControlStateNormal];
```

Всего существует несколько вариантов, определяемых списком констант в iPhone SDK.

```
enum {
   UIControlStateNormal            = 0,
   UIControlStateHighlighted       = 1 << 0,
   UIControlStateDisabled          = 1 << 1,
   UIControlStateSelected          = 1 << 2,
   UIControlStateApplication       = 0x00FF0000,
   UIControlStateReserved          = 0xFF000000
};
```

- Цвет текста.

```
[m_cButton setTitleColor:[UIColor redColor]
           forState:UIControlStateNormal];
```

Цвет задается с помощью объекта типа `UIColor`, как с помощью констант (как в приведенном коде), так и с помощью произвольных компонентов RGB.

- Изображение рядом с текстом.

```
[m_cButton setImage:[UIImage imageNamed:@"1.png"]
           forState:UIControlStateNormal];
```

Синтаксис аналогичен приведенным функциям, но в качестве объекта передается тип `UIImage`. В приведенном примере для создания `UIImage` вызывается статическая функция `imageNamed`, загружающая файл 1.png из папки с программой.

- Фоновое изображение.

```
[m_cButton setBackgroundImage: [UIImage imageNamed:@"2.png"]
           forState:UIControlStateNormal];
```

Для проверки работоспособности была создана программа TestUIButton, код которой приведен в листинге 3.1.

Листинг 3.1. Код программы TestUIButton

Файл TestUIButtonViewController.h:

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

@interface TestUIButtonViewController : UIViewController
{
    IBOutlet UIButton    *m_pTestButton;
```

```
}

@property(retain, nonatomic) UIButton      *m_pTestButton;

- (IBAction) onButtonSetTitle:(id)sender;
- (IBAction) onButtonSetTitleColor:(id)sender;
- (IBAction) onButtonSetImage:(id)sender;
- (IBAction) onButtonSetBackgroundImage:(id)sender;
- (IBAction) onButtonHide:(id)sender;
- (IBAction) onButtonShow:(id)sender;

@end
```

Файл TestUIButtonViewController.m:

```
#import "TestUIButtonViewController.h"

@implementation TestUIButtonViewController
@synthesize m_pTestButton;

// Implement viewDidLoad to do additional setup after loading the view,
// typically from a nib.
- (void)viewDidLoad
{
    [super viewDidLoad];

    // Инициализация псевдослучайных чисел
    srandom(time(NULL));
}

- (IBAction) onButtonSetTitle:(id)sender
{
     [m_pTestButton setTitle:@"setTitle" forState:UIControlStateNormal];
     [m_pTestButton setTitle:@"setTitle (clicked)"
                    forState:UIControlStateHighlighted];
}

- (IBAction) onButtonSetTitleColor:(id)sender
{
     // Компоненты цвета
     int r = random() % 255, g = random() % 255, b = random() % 255;

     [m_pTestButton setTitleColor:[UIColor redColor]
                                forState:UIControlStateHighlighted];
     [m_pTestButton setTitleColor:[UIColor colorWithRed:(float)r/255.0f
                                                  green:(float)g/255.0f
                                                  blue:(float)b/255.0f
                                                  alpha:1]
```

```
                    forState:UIControlStateNormal];
}

- (IBAction) onButtonSetImage:(id)sender
{
      // Button.png должен быть предварительно добавлен
      // в ресурсы программы
      [m_pTestButton setImage: [UIImage imageNamed:@"Button.png"]
                                forState:UIControlStateNormal];
      [m_pTestButton setImage: [UIImage imageNamed:@"Button.png"]
                                forState:UIControlStateHighlighted];
}

- (IBAction) onButtonSetBackgroundImage:(id)sender
{
      [m_pTestButton setBackgroundImage:
                            [UIImage imageNamed:@"BackSky.png"]
                    forState:UIControlStateNormal];
}

- (IBAction) onButtonHide:(id)sender
{
     m_pTestButton.hidden = YES;
}

- (IBAction) onButtonShow:(id)sender
{
     m_pTestButton.hidden = NO;
}

- (void)didReceiveMemoryWarning
{
     // Releases the view if it doesn't have a superview.
     [super didReceiveMemoryWarning];
}

- (void)viewDidUnload
{
}

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

@end
```

Отдельно стоит отметить инициализацию цвета текста случайным цветом, для чего используются функции языка C `random` и `srandom`. Первая функция возвращает случайное число в диапазоне от 0 до `RAND_MAX`, вторая служит для инициализации начальной последовательности псевдослучайных чисел.

Также интересно отметить, что функция `UIColor` для создания цвета использует 4 аргумента (R,G,B + A), таким образом, помимо цвета можно задавать и прозрачность. Диапазон значений цветов задается как число типа `float` в интервале от 0 до 1. Итог показан на рис. 3.3.

Рис. 3.3. Программа демонстрации возможностей `UIButton`

Компонент выбора из нескольких вариантов (*UISegmentedControl*)

Для рассмотрения практического примера добавим в предыдущую программу компонент выбора цвета кнопки из трех заданных вариантов — красного, зеленого и синего. Для этого необходимо выполнить несколько действий.

1. Добавить в класс `TestUIButtonViewController` член типа `UISegmentedControl`.
2. Назначить этот объект как `property`.
3. Создать функцию-обработчик изменения состояния компонента.
4. Добавить компонент на окно и установить связи в Interface Builder.

Результат выполнения первых трех шагов показан в листинге 3.2, новые строки выделены жирным шрифтом.

Листинг 3.2. Модификация класса `TestUIButtonViewController`

Файл TestUIButtonViewController.h:

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

@interface TestUIButtonViewController : UIViewController
{
      IBOutlet UIButton                              *m_pTestButton;
      IBOutlet UISegmentedControl       *m_pSegmentedControl;
}

@property(retain, nonatomic) UIButton      *m_pTestButton;
@property(retain, nonatomic) UISegmentedControl *m_pSegmentedControl;

- (IBAction) onButtonSetTitle:(id)sender;
- (IBAction) onButtonSetTitleColor:(id)sender;
- (IBAction) onButtonSetImage:(id)sender;
- (IBAction) onButtonSetBackgroundImage:(id)sender;
- (IBAction) onButtonHide:(id)sender;
- (IBAction) onButtonShow:(id)sender;
- (IBAction) onSegmentControlIndexChanged;

@end
```

Файл TestUIButtonViewController.m:

```
#import "TestUIButtonViewController.h"

@implementation TestUIButtonViewController
@synthesize m_pTestButton;
@synthesize m_pSegmentedControl;

…
- (IBAction) onSegmentControlIndexChanged
{
      switch (m_pSegmentedControl.selectedSegmentIndex)
      {
            case 0:
                  [m_pTestButton setTitle:@"Red"
                                 forState:UIControlStateNormal];
                  [m_pTestButton setTitleColor:[UIColor redColor]
                                 forState:UIControlStateNormal];
                   break;
            case 1:
                  [m_pTestButton setTitle:@"Green"
                                 forState:UIControlStateNormal];
                  [m_pTestButton setTitleColor:[UIColor greenColor]
```

```
                                forState:UIControlStateNormal];
                    break;
                case 2:
                     [m_pTestButton setTitle:@"Blue"
                                    forState:UIControlStateNormal];
                     [m_pTestButton setTitleColor:[UIColor blueColor]
                                    forState:UIControlStateNormal];
                    break;
        }
}

…

- (void)dealloc
{
        [m_pTestButton release];
        [m_pSegmentedControl release];
        [super dealloc];
}
@end
```

Следующим шагом следует открыть Interface Builder и создать в нем новый компонент Segmented Control, в нем необходимо создать три вкладки: **Red**, **Green** и **Blue**.

Затем с помощью Interface Builder установим связи с созданной переменной `m_pSegmentedControl` и функцией `onSegmentControlIndexChanged`. Если все сделано правильно, контекстное меню компонента должно иметь вид, как на рис. 3.4.

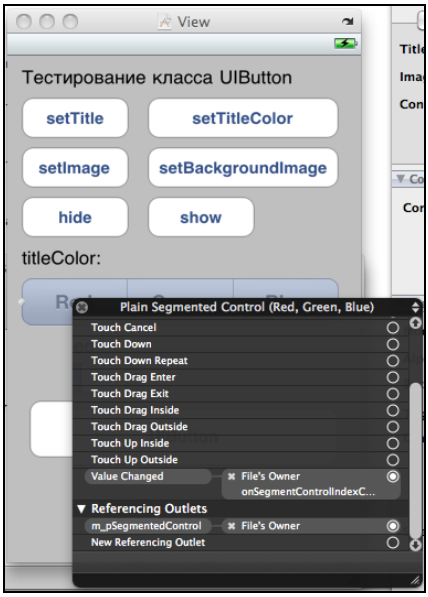


Рис. 3.4. Настройка свойств `UISegmentedControl`

Для переменной мы используем связь Referencing Outlet, для функции используем сообщение **Value Changed**.

Теперь можно переключаться в XCode и запускать программу, и если все сделано правильно, результат будет таким, как на рис. 3.5.

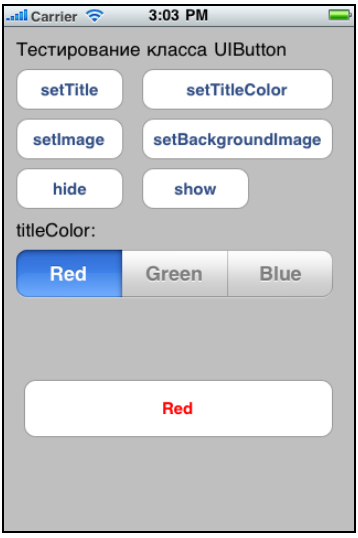


Рис. 3.5. `UISegmentedControl` в программе

Переключатель (*UISwitch*)

Следующим полезным элементом управления является переключатель. Для примера добавим поддержку `UISwitch` в приведенную программу. Для этого модифицируем код, как показано в листинге 3.3.

Листинг 3.3. Добавление `UISwitch` к классу `TestUIButtonViewController`

Файл TestUIButtonViewController.h:

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

@interface TestUIButtonViewController : UIViewController
{
        IBOutlet UIButton            *m_pTestButton;
        IBOutlet UISegmentedControl *m_pSegmentedControl;
        IBOutlet UISwitch            *m_pSwitchControl;
}

@property(retain, nonatomic) UIButton      *m_pTestButton;
@property(retain, nonatomic) UISegmentedControl *m_pSegmentedControl;
```

```
@property(retain, nonatomic) UISwitch      *m_pSwitchControl;

- (IBAction) onButtonSetTitle:(id)sender;
- (IBAction) onButtonSetTitleColor:(id)sender;
- (IBAction) onButtonSetImage:(id)sender;
- (IBAction) onButtonSetBackgroundImage:(id)sender;
- (IBAction) onButtonHide:(id)sender;
- (IBAction) onButtonShow:(id)sender;
- (IBAction) onSegmentControlIndexChanged;
- (IBAction) onSwitchChanged;

@end
```

Файл TestUIButtonViewController.m:

```
#import "TestUIButtonViewController.h"

@implementation TestUIButtonViewController
@synthesize m_pTestButton;
@synthesize m_pSegmentedControl;
@synthesize m_pSwitchControl;

…
- (IBAction) onSwitchChanged
{
       if (m_pSwitchControl.on)
       {
               [m_pTestButton setEnabled:NO];
               [m_pTestButton setTitleColor:[UIColor grayColor]
                              forState:UIControlStateDisabled];
       } else
       {
               [m_pTestButton setEnabled:YES];
       }
}
…
- (void)dealloc
{
        [m_pTestButton release];
        [m_pSegmentedControl release];
        [m_pSwitchControl release];
        [super dealloc];
}

@end
```

Добавим в Interface Builder компонент Switch и установим связи между новым членом класса `m_pSwitchControl` и функцией `onSwitchChanged`. Если все сделано правильно, контекстное меню должно выглядеть, как показано на рис. 3.6.

Теперь при включении Switch Control кнопка должна становиться активной либо неактивной, в зависимости от выбранного состояния. Вид работающей программы показан на рис. 3.7.

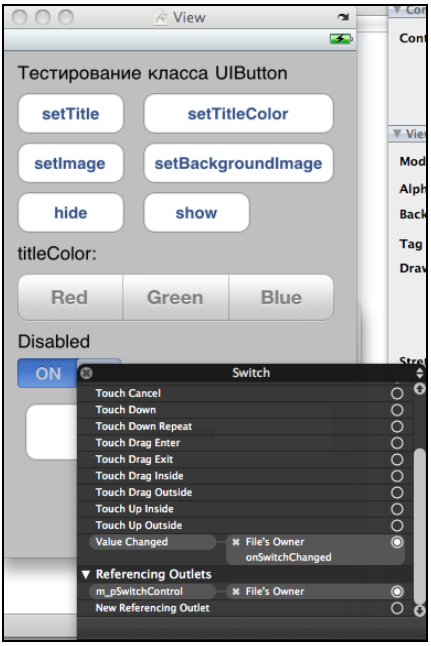


Рис. 3.6. Настройка свойств `UISwitchControl`

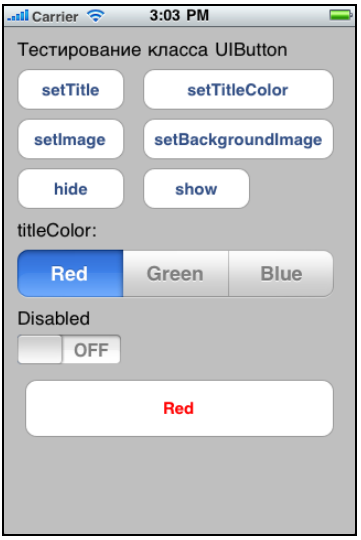


Рис. 3.7. Использование `UISwitchControl`

Мы заодно присваиваем кнопке серый цвет (`grayColor`), чтобы пользователю было видно ее неактивное состояние.

Прокрутка (*UIScrollView*)

Следующим полезным компонентом является `UIScrollView`, позволяющий легко организовать прокрутку содержимого, например изображения или элементов управления. Это может быть актуально для iPhone, где экранная область невелика.

Рассмотрим практический пример — сделаем диалоговое окно, имеющее вертикальную прокрутку. Для этого необходимо сделать следующие шаги.

1. Разместить на окне компонент типа `UIScrollView`, его высоту в настройках можно задать больше высоты экрана.
2. Поверх окна прокрутки разместить кнопку и фотографию, которая без прокрутки не помещается на экране.

3. Добавить в класс элемент типа `UIScrollView` и в функции загрузки `viewDidLoad` указать его свойства.
4. Установить связь через Interface Builder.

На третьем шаге, для того чтобы обеспечить корректную работу компонента, нужно добавить код, показанный в листинге 3.4 жирным шрифтом.

Листинг 3.4. Использование `UIScrollView`

Файл TestUIScrollerViewController.h:

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

@interface TestUIScrollerViewController : UIViewController
{
        IBOutlet UIScrollView                    *m_pScrollView;
}

@property(retain, nonatomic) UIScrollView  *m_pScrollView;

@end
```

Файл TestUIScrollerViewController.m:

```
#import "TestUIScrollerViewController.h"

@implementation TestUIScrollerViewController
@synthesize m_pScrollView;

// Implement viewDidLoad to do additional setup after loading the view,
// typically from a nib.
- (void)viewDidLoad
{
        [super viewDidLoad];

        CGSize s = m_pScrollView.bounds.size;

        int px = self.view.bounds.size.width,
            py = self.view.bounds.size.height;
        [m_pScrollView setBounds:CGRectMake(0, 0, px, py)];
        [m_pScrollView setCenter:CGPointMake(self.view.center.x, py/2)];
        m_pScrollView.contentSize = CGSizeMake(s.width, s.height);
}

- (void)didReceiveMemoryWarning
{
```

```
        // Releases the view if it doesn't have a superview.
         [super didReceiveMemoryWarning];
}

- (void)viewDidUnload
{

}

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

Мы уже стандартным способом добавляем новый член класса. Однако в случае со скроллингом этого недостаточно — компонент будет создан, но прокручиваться не будет. Для того чтобы активизировать прокрутку, необходимо настроить параметры компонента в функции `viewDidLoad`. Для `UIScrollView` задаются размеры видимой области `Bounds`, размеры области содержимого `Content` и центр `Center`. Для получения размеров текущего окна используется свойство `bounds` текущего вида `self.view`.

Связывание с Interface Builder выполняется традиционным способом, и если все сделано нормально, контекстное меню должно иметь вид, как на рис. 3.8.

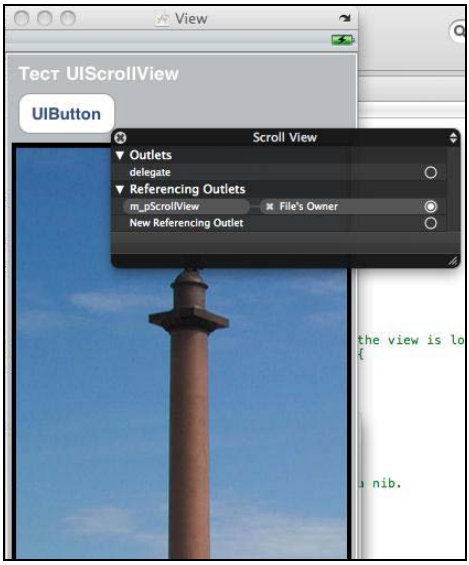


Рис. 3.8. Настройка UIScrollView

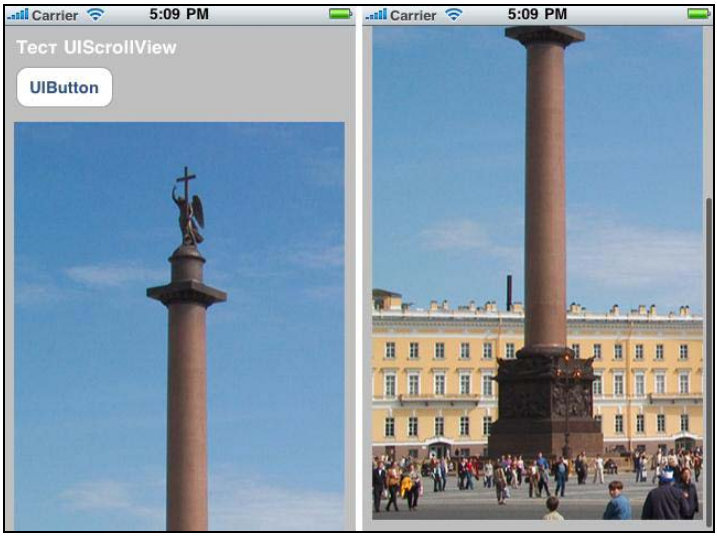


Рис. 3.9. Диалоговое окно со скроллингом содержимого

Окно должно прокручиваться, как показано на рис. 3.9.

Перейдем к не менее актуальной части — вводу и выводу информации.

Ввод и вывод информации — основные компоненты

Ввод текста *UITextField*

Поле ввода текста в iOS называется Text Field. И хотя его использование довольно просто, есть пара интересных моментов.

Рассмотрим программу, которая запрашивает у пользователя число и выводит его в шестнадцатеричном виде. Для создания интерфейса нам понадобится одно поле `UITextField`, одно поле `UILabel` для вывода и кнопка, по нажатию которой будет выполняться преобразование. Создадим класс окна, исходный код которого показан в листинге 3.5. Как и раньше, добавленные функции выделены.

Листинг 3.5. Использование `UITextField`

```
Файл TestUIInputViewController.h:
#import <UIKit/UIKit.h>

@interface TestUIInputViewController : UIViewController
{
     IBOutlet UITextField    *m_pUserInput;
     IBOutlet UILabel          *m_pLabelResult;
```

```
}

- (IBAction) onButtonCalculate:(id)sender;

@property(retain, nonatomic) UITextField   *m_pUserInput;
@property(retain, nonatomic) UILabel              *m_pLabelResult;

@end
```

Файл TestUIInputViewController.m:

```
#import "TestUIInputViewController.h"

@implementation TestUIInputViewController
@synthesize m_pUserInput;
@synthesize m_pLabelResult;

// Implement viewDidLoad to do additional setup after loading the view,
// typically from a nib.
- (void)viewDidLoad
{
      [super viewDidLoad];

      m_pUserInput.text = @"0";
}

- (IBAction) onButtonCalculate:(id)sender
{
     NSString *data = m_pUserInput.text;
     int val = [data intValue];

     m_pLabelResult.text = [NSString stringWithFormat:@"%d = %Xh",
                                                          val, val];
}

- (void)didReceiveMemoryWarning
{
     // Releases the view if it doesn't have a superview.
      [super didReceiveMemoryWarning];
}

- (void)viewDidUnload
{
}

- (void)dealloc
```

```
{
        [m_pUserInput release];
        [m_pLabelResult release];
        [super dealloc];
}

@end
```

Мы создали два объекта: `m_pUserInput` и `m_pLabelResult`, объявили их как `property` и создали функцию нажатия кнопки, в которой выполняются следующие действия:

- текст из поля ввода сохраняется в строке `data` с помощью обращения к свойству `m_pUserInput.text`;
- строка `data` переводится в число с помощью функции `intValue`;
- результат формируется в виде строки с помощью функции `stringWithFormat` и заносится в метку `m_pLabelResult`.

После того как код написан, с помощью Interface Builder нужно связать его с интерфейсом. Отдельно остановимся на свойствах Text Field (рис. 3.10):

- для поля **Keyboard** задано значение **Numbers & Punctuation**, что не позволяет вводить буквы вместо чисел;
- **return Key** — это кнопка, которая отображается в нижней части клавиатуры, ее можно видеть на рис. 3.11. Можно выбрать несколько предустановленных вариантов, например **Go** или **Return**;

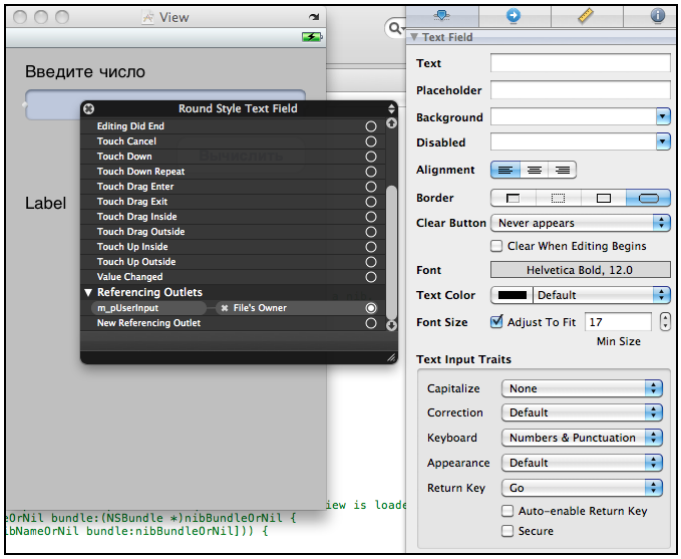


Рис. 3.10. Окно настроек **Text Field**

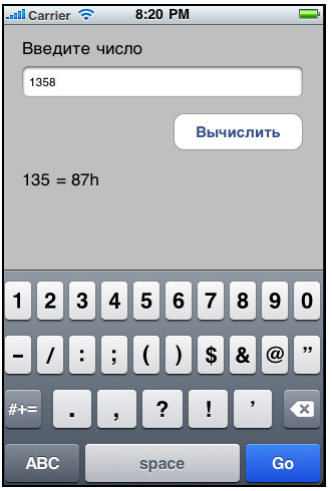


Рис. 3.11. Результат работы программы

- **Clear Button** — опциональный параметр крестика внутри поля ввода, позволяющего быстро очистить компонент от предыдущего текста. Это достаточно удобно, если нужно ввести новое значение. Крестик появляется автоматически, когда фокус ввода находится внутри компонента;
- **Referencing outlet** — привязан к члену класса `m_pUserInput`.

После того как все эти параметры заданы, можно скомпилировать программу. Выявились минимум два ее недостатка:

1. Программа не реагирует на нажатие кнопки **Go**.
2. Лучше выводить результат сразу, во время ввода числа, избавившись от необходимости нажатия кнопки вообще.

Начнем с кнопки **Go**. Для создания обработчика нажатия этой кнопки в протоколе `UITextFieldDelegate` предусмотрена функция `textFieldShouldReturn`. Для этого изменим класс, как показано в листинге 3.6.

Листинг 3.6. Использование `UITextField`, модификация класса

```
- (IBAction) onButtonCalculate:(id)sender
{
     [self Calculate];
}

- (BOOL)textFieldShouldReturn:(UITextField*)textField
{
      [textField resignFirstResponder];
      [self Calculate];
      return YES;
}

- (void) Calculate
{
     NSString *data = m_pUserInput.text;
     int val = [data intValue];

     m_pLabelResult.text = [NSString stringWithFormat:@"%d = %Xh",
                                                    val, val];
}
```

Мы вынесли расчет в отдельную функцию, чтобы избавиться от дублирования кода. Затем вызвали новую функцию из `textFieldShouldReturn`. Отдельно отметим вызов метода `resignFirstResponder`, чтобы убрать экранную клавиатуру.

Скомпилировав код, запускаем программу, вводим число, нажимаем кнопку **Go** — ничего не происходит. Все правильно: для того чтобы наш класс мог получать какие-либо сообщения от элемента управления, этот класс нужно объявить

"делегатом" (представителем) компонента. Для этого нужно в Interface Builder открыть контекстное меню и поле `delegate` привязать к объекту **File's Owner** тем же методом, что описывался ранее. Результат показан на рис. 3.12.

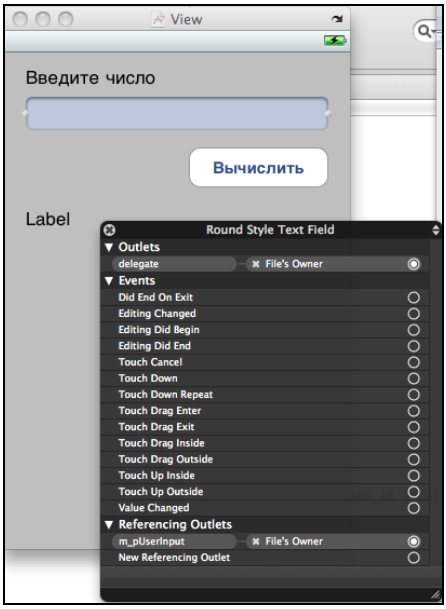


Рис. 3.12. Настройка делегата для компонента

Теперь, после того как класс объявлен как делегат данного компонента, сообщения от компонента будут приходить и обрабатываться корректно, если запустить программу, ввести число и нажать кнопку **Go**. Клавиатура должна исчезнуть, а результат операции появится на экране.

Следующим очевидным улучшением этой программы является вывод результата прямо во время ввода пользователем данных — для этого нужно добавить функцию-обработчик соответствующего сообщения, как показано в листинге 3.7.

Листинг 3.7. Добавление обработчика Editing Changed

Файл TestUIInputViewController.h:

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

@interface TestUIInputViewController : UIViewController
{
        IBOutlet UITextField                *m_pUserInput;
        IBOutlet UILabel                    *m_pLabelResult;
}

- (IBAction) onButtonCalculate:(id)sender;
- (IBAction) onTextChanged:(id)sender;
```

```
- (void) Calculate;

@property(retain, nonatomic) UITextField   *m_pUserInput;
@property(retain, nonatomic) UILabel       *m_pLabelResult;

@end
```

Файл TestUIInputViewController.m:

```
#import "TestUIInputViewController.h"

@implementation TestUIInputViewController
@synthesize m_pUserInput;
@synthesize m_pLabelResult;

// Implement viewDidLoad to do additional setup after loading the view,
// typically from a nib.
- (void)viewDidLoad
{
    [super viewDidLoad];

    m_pUserInput.text = @"0";
    [self Calculate];
}

- (IBAction) onButtonCalculate:(id)sender
{
     [self Calculate];
}

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

     return YES;
}

- (IBAction) onTextChanged:(id)sender
{
     [self Calculate];
}

- (void) Calculate
{
    NSString *data = m_pUserInput.text;
    int val = [data intValue];
```

```
            m_pLabelResult.text = [NSString stringWithFormat:@"%d = %Xh",
                                                    val, val];
}

- (void)didReceiveMemoryWarning
{
    // Releases the view if it doesn't have a superview.
    [super didReceiveMemoryWarning];
}

- (void)viewDidUnload
{
}

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

@end
```

Необходимо привязать новую функцию к сообщению с помощью Interface Builder. Для этого выберем в разделе **Events** сообщение **Editing Changed** и привяжем его к нашему классу. Рядом со значком класса появится имя новой функции `onTextChanged`, а контекстное меню будет иметь вид, как на рис. 3.13.

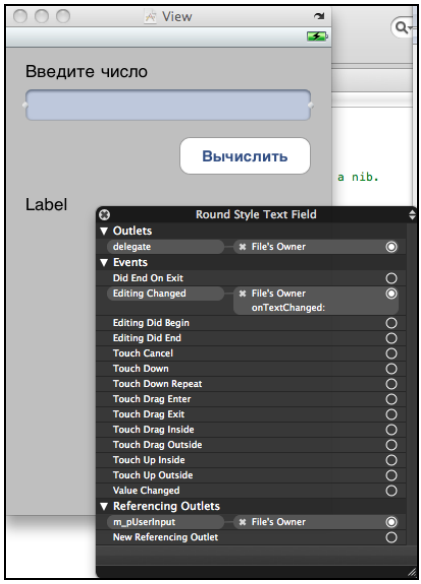


Рис. 3.13. Окончательная настройка компонента

Теперь можно запускать программу, текст в окне результатов должен меняться синхронно с вводом данных пользователем и позволяет вообще отказаться от кнопки **Вычислить**. В более сложных диалоговых окнах такая оптимизация интерфейса позволяет сэкономить место на экране, что для смартфонов весьма актуально.

Ввод аналоговой информации *UISlider*

Для ввода различных данных с помощью "ползунка" в iOS существует класс `UISlider`. Добавим в приведенную программу настройку прозрачности результатов вывода и будем регулировать прозрачность компонента в интервале 0−1. Для таких действий `UISlider` как раз подходит лучше всего.

Нужно выполнить уже знакомые три шага: добавить переменную для нового компонента, добавить компонент с помощью Interface Builder и установить связи между ними. Первый шаг показан в листинге 3.8, новые строки выделены жирным шрифтом.

Листинг 3.8. Добавление компонента `UISlider`

Файл TestUIInputViewController.h:

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

@interface TestUIInputViewController : UIViewController
{
      IBOutlet  UITextField         *m_pUserInput;
      IBOutlet  UILabel             *m_pLabelResult;
      IBOutlet UISlider             *m_pSlider;
}

- (IBAction) onButtonCalculate:(id)sender;
- (IBAction) onTextChanged:(id)sender;
- (IBAction) onSliderChanged:(id)sender;
- (void) Calculate;

@property(retain, nonatomic) UITextField   *m_pUserInput;
@property(retain, nonatomic) UILabel              *m_pLabelResult;
@property(retain, nonatomic) UISlider             *m_pSlider;

@end
```

Файл TestUIInputViewController.m:

```
#import "TestUIInputViewController.h"

@implementation TestUIInputViewController
@synthesize m_pUserInput;
@synthesize m_pLabelResult;
```

```
@synthesize m_pSlider;

// Implement viewDidLoad to do additional setup after loading the view,
// typically from a nib.
- (void)viewDidLoad
{
       [super viewDidLoad];

       m_pUserInput.text = @"0";
       [self Calculate];
}

- (IBAction) onButtonCalculate:(id)sender
{
       [self Calculate];
}

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

       return YES;
}

- (IBAction) onTextChanged:(id)sender
{
       [self Calculate];
}

- (IBAction) onSliderChanged:(id)sender
{
      float alpha = m_pSlider.value;
      m_pLabelResult.alpha = alpha;
}

- (void) Calculate
{
      NSString *data = m_pUserInput.text;
      int val = [data intValue];

      m_pLabelResult.text = [NSString stringWithFormat:@"%d = %Xh",
                                                        val, val];
}
```

```
- (void)didReceiveMemoryWarning
{
        // Releases the view if it doesn't have a superview.
        [super didReceiveMemoryWarning];
}

- (void)viewDidUnload
{
}

- (void)dealloc
{
        [m_pUserInput release];
        [m_pLabelResult release];
        [m_pSlider release];
        [super dealloc];
}

@end
```

Мы добавили в исходный код член класса `UISlider` и функцию-обработчик `onSliderChanged`. Следующим шагом нужно добавить компонент через Interface Builder и установить связи, как показано на рис. 3.14. Необходимо привязать переменную `UISlider` в разделе **Referencing Outlet** и функцию `onSliderChanged` в разделе **Events**.

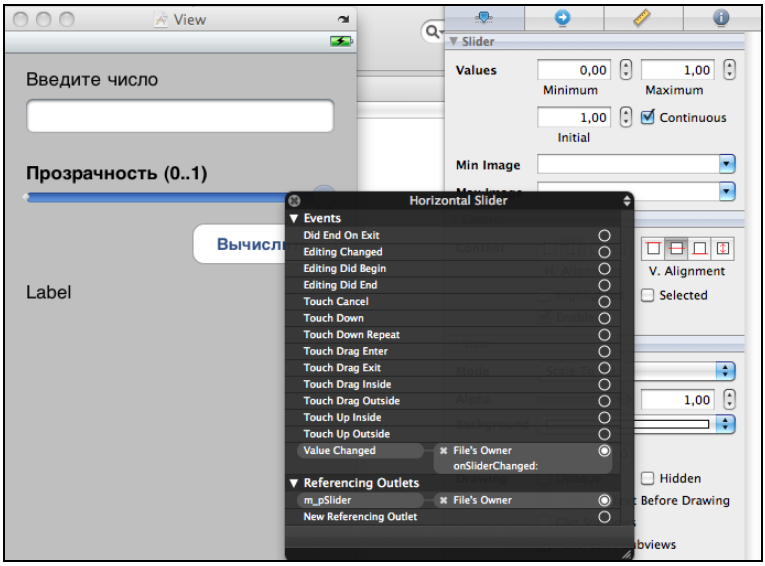


Рис. 3.14. Настройка компонента `UISlider`

Стоит также обратить внимание на настройки `UISlider`, показанные в правой части окна. Как можно видеть, в разделе **Values** (Значения) задаются поля **Minimum** и **Maximum**, которые по умолчанию инициализируются как **0,00** и **1,00** соответственно. Такой диапазон нас как раз устраивает, он соответствует значениям прозрачности в iOS. Для доступа к текущему положению ползунка используется системное свойство `value`, изменение прозрачности элемента `UILabel` производится с помощью стандартного свойства `alpha`.

Когда компонент создан и связи установлены, можно запускать программу. Изменение положения ползунка слайдера должно изменять прозрачность компонента, как видно из рис. 3.15.

Поскольку при запуске программы компонент непрозрачен, в поле **Initial** устанавливается **1.0**. При желании компонент `UISlider` можно "украсить", задав значки для минимальных и максимальных значений, пример такого использования часто можно видеть в системных настройках яркости или контрастности.

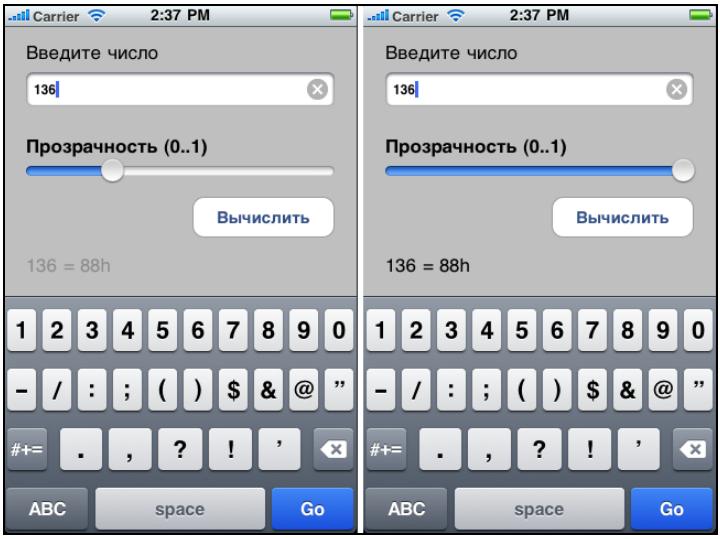


Рис. 3.15. Использование `UISlider` в программе

Ввод информации из списка (*UIPickerView*)

Элемент ввода из предустановленного списка значений можно видеть в диалоговом окне ввода даты. Его принципиальной особенностью является невозможность ввести некорректные данные, что отличает `UIPickerView` от других способов ввода. Важно заметить, что сам компонент не содержит никаких данных, он лишь запрашивает класс-родитель, который, в свою очередь, обязан предоставить ему данные.

Рассмотрим использование `UIPickerView` на примере вывода чисел, добавим в предыдущую программу список предустановленных значений (рис. 3.16) в три шага.

1. Добавим член класса `UIPickerView`, как показано в листинге 3.9.
2. Добавим компонент с помощью Interface Builder и установим связи.
3. Добавим в исходный код функции предоставления данных.

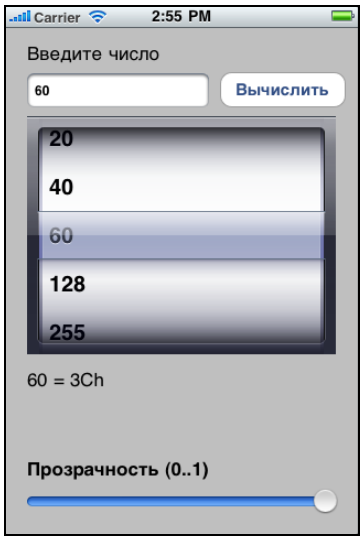


Рис. 3.16. Использование `UIPickerView` в программе

Листинг 3.9. Добавление компонента `UIPickerView`

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

@interface TestUIInputViewController :
            UIViewController<UIPickerViewDataSource,UIPickerViewDelegate>
{
       IBOutlet UITextField         *m_pUserInput;
       IBOutlet UILabel             *m_pLabelResult;
       IBOutlet UISlider            *m_pSlider;
       IBOutlet UIPickerView        *m_pPickerView;
}

- (IBAction) onButtonCalculate:(id)sender;
- (IBAction) onTextChanged:(id)sender;
- (IBAction) onSliderChanged:(id)sender;
- (void) Calculate;

@property(retain, nonatomic) UITextField   *m_pUserInput;
```

```
@property(retain, nonatomic) UIPickerView  *m_pPickerView;
@property(retain, nonatomic) UILabel       *m_pLabelResult;
@property(retain, nonatomic) UISlider      *m_pSlider;

@end
```

Методы `@synthesize` и вызов `release` в файле TestUIInputViewController.m показывать не будем, они аналогичны предыдущему примеру. Отметим строки `UIPickerViewDataSource` и `UIPickerViewDelegate`, которые указывают компилятору, что класс должен реализовать методы этих протоколов.

Вторым шагом необходимо добавить `PickerView` в диалоговое окно и установить связи, как показано на рис. 3.17.

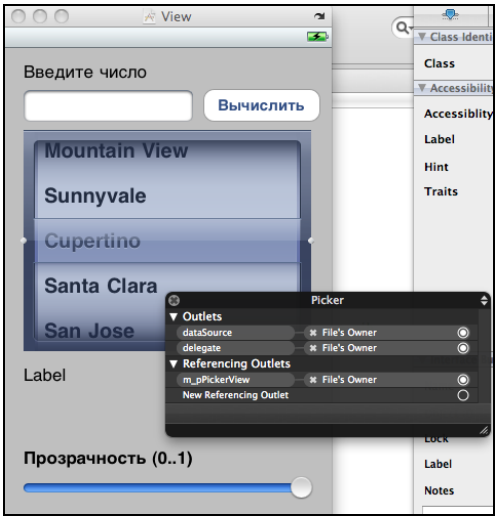


Рис. 3.17. Настройка компонента `UIPickerView`

Мы связали член класса `m_pPickerView` с компонентом и указали, что источником данных (`dataSource`) и делегатом (`delegate`) будет наш класс. Теперь можно скомпилировать и запустить программу. Однако если мы это сделаем, то увидим, что список пуст. Необходимо создать функции, обеспечивающие компонент данными. Как можно видеть из листинга 3.9, мы использовали два новых элемента — `UIPickerViewDataSource` и `UIPickerViewDelegate`. Из документации Apple следует, что протокол `UIPickerViewDataSource` содержит две функции:

- `numberOfComponentsInPickerView` — функция возвращает количество столбцов;
- `pickerView:numberOfRowsInComponent` — функция возвращает количество строк в каждом столбце.

Протокол `UIPickerViewDelegate` имеет довольно-таки много функций, из которых нас будут интересовать основные две:

- `pickerView:titleForRow:forComponent` — возвращает имя для компонента;
- `pickerView:didSelectRow` — вызывается при выборе компонента пользователем.

Мы будем использовать простейший вариант, в котором данные непосредственно хранятся в коде. Это не лучший вариант с точки зрения идеологии программирования, но для данного примера вполне достаточный. Пример реализации класса `TestUIInputViewController` с добавленными функциями представлен в листинге 3.10.

Листинг 3.10. Реализация доступа к данным для `UIPickerView`

```
#import "TestUIInputViewController.h"

@implementation TestUIInputViewController
@synthesize m_pUserInput;
@synthesize m_pPickerView;
@synthesize m_pLabelResult;
@synthesize m_pSlider;

// Implement viewDidLoad to do additional setup after loading the view,
// typically from a nib.
- (void)viewDidLoad
{
    [super viewDidLoad];

    m_pUserInput.text = @"0";
    [self Calculate];
}

- (IBAction) onButtonCalculate:(id)sender
{
     [self Calculate];
}

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

     return YES;
}

- (IBAction) onTextChanged:(id)sender
{
     [self Calculate];
}

- (IBAction) onSliderChanged:(id)sender
{
```

```
    float alpha = m_pSlider.value;
    m_pLabelResult.alpha = alpha;
}

- (NSInteger)numberOfComponentsInPickerView:(UIPickerView*)thePickerView
{
    return 1;
}

-  (NSInteger)pickerView:(UIPickerView*)thePickerView
                  numberOfRowsInComponent:(NSInteger)component
{
    return 10;
}

- (NSString*)pickerView:(UIPickerView*)thePickerView
                    titleForRow:(NSInteger)row
                    forComponent:(NSInteger)component
{
    NSArray *data = [NSArray arrayWithObjects:@"0", @"1", @"2", @"10",
                                        @"20", @"40", @"60", @"128",
                                        @"255",  @"512", nil];

    return [data objectAtIndex:row];
}

- (void)pickerView:(UIPickerView *)thePickerView
                          didSelectRow:(NSInteger)row
                          inComponent:(NSInteger)component
{
     NSString *text = [self pickerView:m_pPickerView
                               titleForRow:row
                               forComponent:component];

     m_pUserInput.text = text;
     [self Calculate];
}

- (void) Calculate
{
    NSString *data = m_pUserInput.text;
    int val = [data intValue];
    m_pLabelResult.text = [NSString stringWithFormat:@"%d = %Xh",
                                                    val, val];
```

```
}


- (void)didReceiveMemoryWarning
{
    // Releases the view if it doesn't have a superview.
    [super didReceiveMemoryWarning];
}

- (void)viewDidUnload
{
}

- (void)dealloc
{
     [m_pUserInput release];
     [m_pPickerView release];
     [m_pLabelResult release];
     [m_pSlider release];
     [super dealloc];
}

@end
```

Суть реализации довольно проста. Функция `numberOfComponents` (количество секций) всегда возвращает 1, функция `numberOfRowsInComponent` возвращает 10, функция `titleForRow` возвращает строку из массива по индексу, который может быть не более 10. Функция `didSelectRow` вызывается в тот момент, когда пользователь изменил текущий элемент в списке, для того чтобы не дублировать код, а для получения текущей строки мы самостоятельно вызываем функцию `titleForRow` еще раз, используя полученный текст для вывода результата. Если все сделано правильно, список должен быть наполнен значениями массива, хранящегося в функции `titleForRow`, как можно видеть на рис. 3.16.

Вывод информации в табличной форме

Как и `UIPickerView` класс `UITableView` не хранит в себе данные, а лишь запрашивает их у класса-родителя.

Рассмотрим пример интерфейса на рис. 3.2. Реализуем класс таблицы, отображающий список планет солнечной системы. Наш класс должен содержать методы класса `UITableViewDataSource` для источников данных и методы класса `UITableViewDelegate` для получения сообщений от компонента.

Полное описание протокола `UITableViewDataSource` можно найти в документации Apple, в разделе "UITableViewDataSource Protocol Reference". Нам для обеспе-

чения вывода табличных данных достаточно реализовать следующий минимальный набор функций:

- `numberOfSectionsInTableView` — таблица может быть разделена на секции, в нашем случае количество секций равно единице;
- `numberOfRowsInSection` — количество строк в секции. В нашем примере число элементов таблицы жестко задано;
- `cellForRowAtIndexPath` — функция должна возвращать объект типа `UITableViewCell`. В iOS ячейка таблицы это отдельный объект достаточно сложной структуры (позволяющий хранить не только текст, но и изображение, другие компоненты, использовать разные шрифты);
- `willSelectRowAtIndexPath` — функция вызывается перед тем, как новый объект будет выделен пользователем. Если выделение объектов в таблицы в программе не предусмотрено, эта функция может возвращать `nil`;
- `didSelectRowAtIndexPath` — функция вызывается после того, как пользователь выбрал строку таблицы. Здесь может быть обработка каких-либо действий, например открытие окна с более подробным описанием объекта.

Нам необходимо выполнить следующую последовательность шагов.

1. Добавить в класс элемент типа `UITableView` и описать его как `property`.
2. Добавить таблицу через Interface Builder и установить связи.
3. Добавить в класс описанные функции.

Рассмотрим код готового класса в листинге 3.11.

Листинг 3.11. Класс с поддержкой вывода табличных данных

Файл TestUITableViewController.h

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

@interface TestUITableViewController : UIViewController<UITableViewDelegate,

UITableViewDataSource>
{
    IBOutlet UITableView    *m_pItemsTable;
}

@property(retain, nonatomic) UITableView   *m_pItemsTable;

@end
```

Файл TestUITableViewController.m

```
#import "TestControlsViewController.h"

@implementation TestUITableViewController
@synthesize m_pItemsTable;
```

```
-(NSInteger)numberOfSectionsInTableView:(UITableView *)tableView
{
    return 1;
}

-(NSInteger)tableView:(UITableView *)tableView
                                          numberOfRowsInSec-
tion:(NSInteger)section
{
    return 8;
}

- (UITableViewCell *)tableView:(UITableView *)tableView
                    cellForRowAtIndexPath:(NSIndexPath *)indexPath
{

    static NSString *MyIdentifier = @"IDDQD"; // any ID

    UITableViewCell *cell = [m_pItemsTable
                    dequeueReusableCellWithIdentifier:MyIdentifier];

    // If no cell is available, create a new one
    // using the given identifier.
    if (cell == nil)
    {
        // Use the default cell style.
        cell = [[[UITableViewCell alloc]
                    initWithStyle:UITableViewCellStyleDefault
                    reuseIdentifier:MyIdentifier] autorelease];
    }

    NSArray *temp = [[NSArray alloc] initWithObjects:@"Меркурий",
                              @"Венера", @"Земля", @"Марс", @"Юпитер",
                              @"Сатурн", @"Уран", @"Нептун", nil];

    // Set text
    cell.textLabel.text = [temp objectAtIndex:indexPath.row];

    [temp release];

    return cell;
}

- (NSIndexPath *)tableView:(UITableView *)tableView
                    willSelectRowAtIndexPath:(NSIndexPath *)indexPath
```

```
{
     return indexPath; //return nil;
}

- (void)tableView:(UITableView*)tableView
                      didSelectRowAtIndexPath:(NSIndexPath *)indexPath
{
     UIAlertView *simpleAlert = [[UIAlertView alloc]
                                 initWithTitle:@"Объект был выбран"
                                 message:@"..."
                                 delegate:self
                                 cancelButtonTitle:@"OK"
                                 otherButtonTitles:nil,
                                 nil];
      [simpleAlert show];
      [simpleAlert release];
}


- (void)didReceiveMemoryWarning
{
     // Releases the view if it doesn't have a superview.
    [super didReceiveMemoryWarning];
}

- (void)viewDidUnload
{
}

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

@end
```

Количество элементов и их содержимое фиксировано и прописано прямо в исходном коде. Это не лучший вариант с точки зрения стиля программирования, но вполне допустимый для простого примера. В настоящих проектах графический интерфейс и данные все-таки лучше по возможности разделять.

Отдельно стоит рассмотреть функцию `cellForRowAtIndexPath`: она запрашивает у таблицы объект с уникальным идентификатором, а если такого объекта нет, создает его. Далее, когда объект создан, с помощью системного свойства `textLabel` задается его текст.

Следующим шагом необходимо привязать созданный класс к интерфейсу с помощью Interface Builder. Как и ранее, необходимо привязать не только имя созданной переменной, но и объекты `delegate` и `datasource`, как показано на рис. 3.18.

Если все было сделано правильно, интерфейс программы будет похож на рис. 3.19.

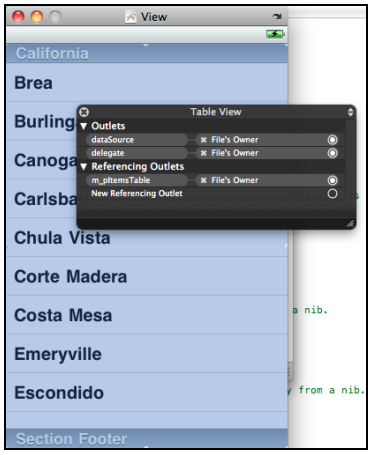


Рис. 3.18. Настройка компонента UITableView

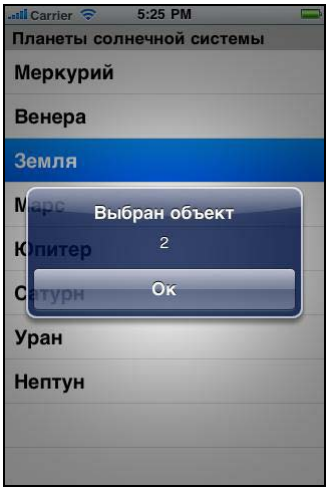


Рис. 3.19. `UITableView` в программе

В этом компоненте все параметры (вид, прозрачность, фон и пр.) заданы по умолчанию.

Некоторые читатели могут задаться вопросом, почему объектов в таблице 8, а не 9: по последним астрономическим данным Плутон является карликовой планетой и не относится к "полноценным" планетам солнечной системы, по новой классификации — это лишь один из объектов пояса Койпера.

Вывод предупреждений

Вывод предупреждений обеспечен в iOS отдельным классом `UIAlertView`, облегчающим программисту эту задачу.

Простое сообщение

Простейший пример вызова `UIAlertView` выглядит так, как показано в листинге 3.12.

Листинг 3.12. Пример `UIAlertView`

```
UIAlertView *simpleAlert = [[UIAlertView alloc]
      initWithTitle:@"Предупреждение"
      message:@"Нажмите кнопку выбора действия"
      delegate:nil
```

```
      cancelButtonTitle:@"OK"
      otherButtonTitles:nil, nil];
[simpleAlert show];
[simpleAlert release];
```

Пользователь может настроить все основные параметры: заголовок окна, сообщение, текст кнопки и т. д.

Результат выполнения этого кода можно видеть на рис. 3.20.

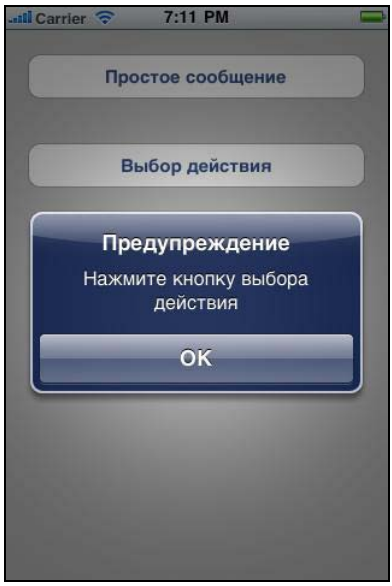


Рис. 3.20. Вывод сообщения `UIAlertView`

Операционная система автоматически затеняет цвет основного окна, и само окно сообщения является полупрозрачным, чтобы видеть выводимые в главном окне данные.

Сообщение с запросом

Следующим примером выведем сообщение с двумя кнопками, его можно использовать для запроса подтверждений у пользователя (листинг 3.13).

Листинг 3.13. `UIAlertView` с запросом

```
UIAlertView *multiAlert = [[UIAlertView alloc]
     initWithTitle:@"Предупреждение"
     message:@"Выполнить действие?" delegate:self
     cancelButtonTitle:@"Нет" otherButtonTitles:@"Да", nil];
 [multiAlert show];
 [multiAlert release];
```

В параметре `otherButtonTitles` может передаваться список кнопок (рис. 3.21). А как узнать, какую кнопку нажал пользователь? Для ответа на этот вопрос присмотримся к вызову функции, точнее к строке `delegate:self`. Действительно, с помощью указания делегата мы получаем обработку сообщений окна `UIAlertView`.

Далее необходимо реализовать функцию `alertView:clickedButtonAtIndex` и производить нужные действия в соответствии с логикой программы.

Следующим примером рассмотрим сообщение с несколькими кнопками, вид которого показан на рис. 3.22.

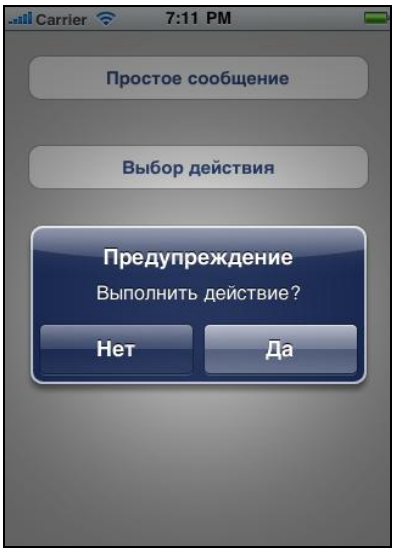


Рис. 3.21. Вывод запроса с помощью UIAlertView

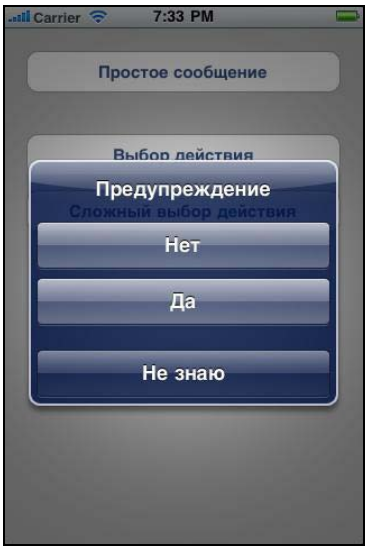


Рис. 3.22. Запрос с помощью UIAlertView

Для создания такого сообщения мы используем код, показанный в листинге 3.14.

Листинг 3.14. `UIAlertView` с запросом

```
UIAlertView *multiAlert = [[UIAlertView alloc] init];
[multiAlert setDelegate:self];
[multiAlert setTitle:@"Предупреждение"];
[multiAlert addButtonWithTitle:@"Нет"];
[multiAlert addButtonWithTitle:@"Да"];
[multiAlert addButtonWithTitle:@"Не знаю"];
[multiAlert show];
[multiAlert release];
```

Вместо функции `initWithTitle` с множеством параметров мы используем обычную функцию `init`, а все требуемые данные задаем отдельно.

Как обрабатывать сообщения в том случае, если в программе используется несколько разных сообщений? Для этого в `UIAlertView` существует опциональное поле `tag`, в котором можно хранить соответствующий идентификатор (листинг 3.15).

Листинг 3.15. `UIAlertView` с запросом

Файл TestUIAlertViewController.h

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

@interface TestUIAlertViewController : UIViewController
{
      IBOutlet UILabel *m_pResults;
}

- (IBAction) onSimpleMessage:(id)sender;
- (IBAction) onMultiMessage:(id)sender;
- (IBAction) onMultiMessage3:(id)sender;

@property(retain, nonatomic) UILabel *m_pResults;

@end
```

Файл TestUIAlertViewController.m

```
#import "TestUIAlertViewController.h"

@implementation TestUIAlertViewController
@synthesize m_pResults;

// Implement viewDidLoad to do additional setup after loading the view,
// typically from a nib.
- (void)viewDidLoad
{
    [super viewDidLoad];
    m_pResults.text = @"";
}

- (IBAction) onSimpleMessage:(id)sender
{
     UIAlertView *simpleAlert = [[UIAlertView alloc]
           initWithTitle:@"Предупреждение"
           message:@"Нажмите кнопку выбора действия"
           delegate:nil
           cancelButtonTitle:@"OK"
           otherButtonTitles:nil, nil];
      [simpleAlert show];
      [simpleAlert release];
}

- (IBAction) onMultiMessage:(id)sender
```

```
{
      UIAlertView *multiAlert = [[UIAlertView alloc]
                initWithTitle:@"Предупреждение"
                message:@"Выполнить действие?" delegate:self
                cancelButtonTitle:@"Нет" otherButtonTitles:@"Да", nil];
      multiAlert.tag = 123;
      [multiAlert show];
      [multiAlert release];
}

- (IBAction) onMultiMessage3:(id)sender
{
      UIAlertView *multiAlert = [[UIAlertView alloc] init];
      [multiAlert setDelegate:self];
      [multiAlert setTitle:@"Предупреждение"];
      [multiAlert addButtonWithTitle:@"Нет"];
      [multiAlert addButtonWithTitle:@"Да"];
      [multiAlert addButtonWithTitle:@"Не знаю"];
      multiAlert.tag = 124;
      [multiAlert show];
      [multiAlert release];
}

- (void)alertView:(UIAlertView *)alertView
             clickedButtonAtIndex:(NSInteger)buttonIndex
{
      if (alertView.tag == 123)
      {
            if (buttonIndex == 0)
                  m_pResults.text = @"Вы выбрали Нет";
            if (buttonIndex == 1)
                  m_pResults.text = @"Вы выбрали Да";
      }
      if (alertView.tag == 124)
      {
            if (buttonIndex == 0)
                  m_pResults.text = @"Вы выбрали Нет";
            if (buttonIndex == 1)
                  m_pResults.text = @"Вы выбрали Да";
            if (buttonIndex == 2)
                  m_pResults.text = @"Вы выбрали Не знаю";
      }
}

- (void)didReceiveMemoryWarning
{
      // Releases the view if it doesn't have a superview.
      [super didReceiveMemoryWarning];
```

```
}

- (void)viewDidUnload
{
}

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

@end
```

Код описывает три кнопки для активации трех различных типов сообщений и поле `UILabel`, в которое выводится результат выполнения. Настройки Interface Builder для элементов интерфейса не показываются, они аналогичны описанным в предыдущих разделах.

Вывод графических изображений

Рассмотрим программу, выводящую фотографии различных времен года. В программе используются два основных компонента — `UISegmentedControl` для переключения и `UIImageView` для отображения картинки. Изображения времен года будут храниться в программе локально, а по нажатию отдельной кнопки изображение будет загружаться из сети Интернет.

Рассмотрим текст программы, показанный в листинге 3.16.

Листинг 3.16. Класс для вывода с помощью `UIImageView`

Файл TestUIImageViewController.h:

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

@interface TestUIImageViewController : UIViewController
{
        IBOutlet UISegmentedControl                *m_pSeasons;
        IBOutlet UIImageView                       *m_pImageView;
}

@property(retain, nonatomic) UISegmentedControl    *m_pSeasons;
@property(retain, nonatomic) UIImageView           *m_pImageView;

- (IBAction) onSeasonsChanged:(id)sender;
- (IBAction) onLoadWeb:(id)sender;

@end
```

Файл TestUIImageViewController.m:

```
#import "TestUIImageViewController.h"

@implementation TestUIImageViewController
@synthesize m_pSeasons;
@synthesize m_pImageView;

// Implement viewDidLoad to do additional setup after loading the view,
// typically from a nib.
- (void)viewDidLoad
{
    [super viewDidLoad];
}

- (IBAction) onSeasonsChanged:(id)sender
{
    int sel_index = m_pSeasons.selectedSegmentIndex;

    switch (sel_index)
    {
    case 0:
        m_pImageView.image = [UIImage imageNamed: @"IMG_0.JPG"];
        break;
    case 1:
        m_pImageView.image = [UIImage imageNamed: @"IMG_1.JPG"];
        break;
    case 2:
        m_pImageView.image = [UIImage imageNamed: @"IMG_2.JPG"];
        break;
    case 3:
        m_pImageView.image = [UIImage imageNamed: @"IMG_3.JPG"];
        break;
    }
}

- (IBAction) onLoadWeb:(id)sender
{
    NSURL   *imageURL = [NSURL URLWithString:
                @"http://www.google.ru/images/srpr/nav_logo27.png"];
    NSData  *imageData = [NSData dataWithContentsOfURL:imageURL];
    UIImage *image = [UIImage imageWithData:imageData];

    m_pImageView.image = image;
```

```
    m_pSeasons.selectedSegmentIndex = -1;
}

- (void)didReceiveMemoryWarning
{
    // Releases the view if it doesn't have a superview.
    [super didReceiveMemoryWarning];
}
- (void)viewDidUnload
{
}
- (void)dealloc
{
    [m_pSeasons release];
    [m_pImageView release];
    [super dealloc];
}
@end
```

Разместим в окне программы элементы **Segmented Control** и **Image View**: в настройках свойств создадим четыре элемента, соответствующие временам года. Установим связи с членом класса `m_pSeasons` и функцией `onSeasonsChanged`, как показано на рис. 3.23. Также установим связь `UIImageView` с членом класса `m_pImageView`.

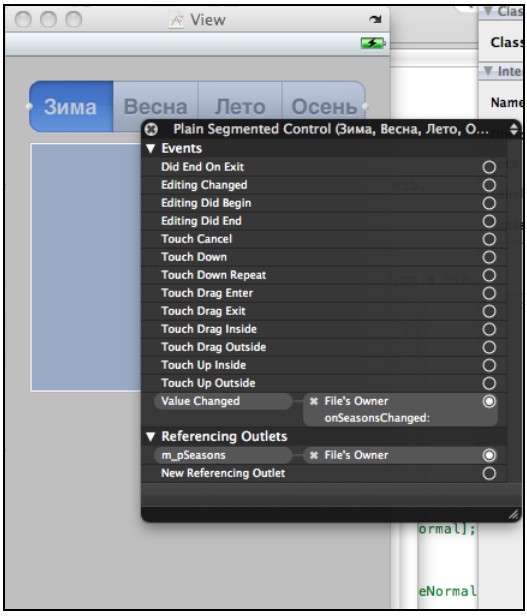


Рис. 3.23. Настройка компонента **Segmented Control**

Как можно видеть, загрузка изображений выполняется в функции `onSeasonsChanged`, для загрузки мы используем статическую функцию `UIImage::imageNamed`, возвращающую объект `UIImage`. Для загрузки изображения непосредственно на экран используется системное свойство `image` класса `UIImageView`. Естественно, что нужное изображение предварительно должно быть загружено в ресурсы программы.

Следующим шагом является загрузка изображений из сети Интернет функцией `onLoadWeb`. Загрузка изображения выполняется в два этапа: получение данных из сети с помощью функции `dataWithContentsOfURL` и формирования изображения с помощью функции `imageWithData`. Отметим, что при большом размере загружаемого изображения программа "зависнет" и перестанет реагировать на действия пользователя до тех пор, пока данные не загрузятся. В реальных проектах этого следует избегать и использовать альтернативные способы: либо асинхронную загрузку данных (этот способ будет описан далее), либо выделение кода загрузки в отдельный поток.

Если все было сделано правильно, вид программы в разных режимах должен соответствовать рис. 3.24.

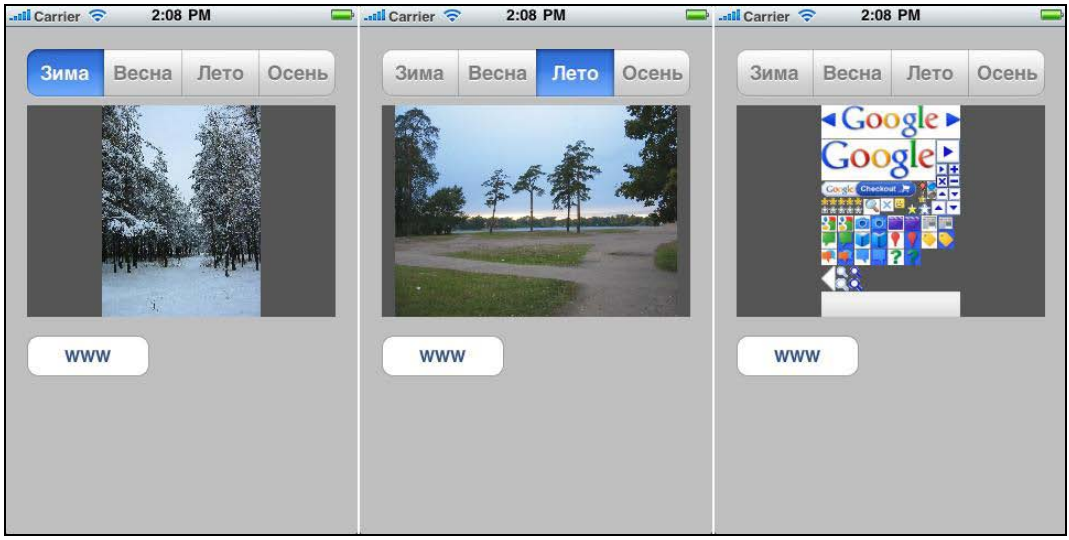


Рис. 3.24. Вывод изображений с помощью `UIImageView`

На этом мы закончим с основами ввода и вывода данных в iOS и перейдем к отображению данных более сложной структуры.

Глава 4

Форматирование и вывод табличных данных

Перед тем как перейти к более сложным методам вывода данных, стоит разобраться с многооконным представлением данных на мобильных устройствах. В iOS каждое окно выводится на весь экран, поэтому единственным способом представления данных является их последовательное открытие. Например, в почтовой программе первым окном может являться список ящиков, при выборе ящика вторым окном пользователю предлагается выбрать папку (Входящие, Исходящие и др.), в третьем окне выводится список писем. Рассмотрим реализацию подобного механизма подробнее.

Иерархическое представление интерфейса программы

Предыдущие программы в данной книге имели одно основное окно. В любой из программ для обеспечения функционирования окна в iOS требуется два основных компонента:

- класс, обеспечивающий функционирование окна, обычно наследуемый от `UIViewController`;
- XIB-файл, хранящий ресурсы окна и связи между интерфейсом и кодом программы.

Очевидно, что для создания нового окна необходимо сделать то же самое. Создадим программу с названием TestUIViews и добавим в нее первое окно **First View** — создадим новый класс, как показано на рис. 4.1. Выберем класс, наследуемый от `UIViewController`, и не забудем установить флажок **With XIB for user interface.** В качестве имени нового класса введем `FirstView`. После этого XCode создаст три файла: FirstView.h, FirstView.m и FirstView.xib. Фактически новое окно создано, и мы можем заполнять его содержимым. Но сначала необходимо вывести его на экран. Добавим в класс основного окна код, позволяющий активизировать наше новое окно.

Создадим в основном окне кнопку и свяжем ее с обработчиком `onShowFirst` (листинг 4.1). Вначале создается объект нужного типа, для чего используется функция `initWithNibName` класса `UIViewController`.

Параметрами этой функции являются имя ресурса (в нашем случае `FirstView`) и опциональное поле `bundle`, которое в нашем случае не используется.

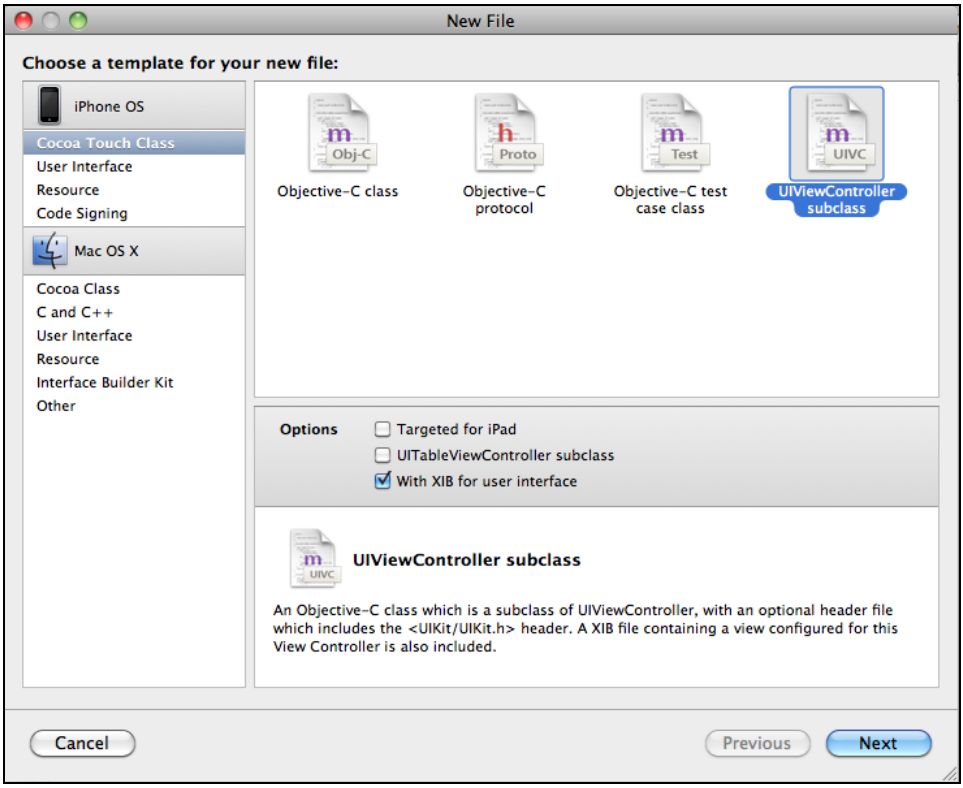


Рис. 4.1. Создание нового класса в XCode

Листинг 4.1. Активизация созданного окна `FirstView`

```
- (IBAction) onShowFirst:(id)sender
{
   FirstView *wnd1 =  [[FirstView alloc] initWithNibName:@"FirstView"
                                         bundle:nil];
   [self presentModalViewController:wnd1 animated:YES];
   [wnd1 release];
}
```

Объект создан, активируем его с помощью функции `presentModalViewController` класса `UIViewController`. Интересно отметить наличие слова `animated`: iOS автоматически добавляет анимацию при создании и закрытии окна. С помощью дополнительных параметров можно настраивать параметры анимации, например скорость или способ появления окна на экране (сбоку, снизу).

Данного кода уже достаточно, чтобы вывести окно на экран, но не менее важно дать возможность пользователю вернуться обратно. Для этого используется функция `dismissModalViewControllerAnimated`, которая закрывает окно и возвращает фокус ввода. Рассмотрим код класса `FirstView` в листинге 4.2.

Листинг 4.2. Класс окна `FirstView`

Файл FirstView.h:

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

@interface FirstView : UIViewController
{

}

- (IBAction) onButtonBack:(id)sender;

@end
```

Файл FirstView.m:

```
#import "FirstView.h"

@implementation FirstView

// Implement viewDidLoad to do additional setup after loading the view,
// typically from a nib.
- (void)viewDidLoad
{
    [super viewDidLoad];
}

- (IBAction) onButtonBack:(id)sender
{
     [self dismissModalViewControllerAnimated:YES];
}

- (void)didReceiveMemoryWarning
{
    // Releases the view if it doesn't have a superview.
    [super didReceiveMemoryWarning];
}

- (void)viewDidUnload
{
    [super viewDidUnload];
}


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

@end
```

Код практически не отличается от того, что приводился ранее, за исключением функции `onButtonBack`, которая выполняет закрытие окна.

Следующим шагом необходимо добавить кнопку возврата в интерфейс окна `FirstView`. Можно просто поместить кнопку в окне, но это выглядит не очень красиво. Поместим в окне элемент **Navigation Bar** и на нем кнопку **Navigation Item**. Установим связи кнопки и обработчика `onButtonBack` с помощью Interface Builder. Теперь, если запустить программу, окна будут переключаться, как показано на рис. 4.2.

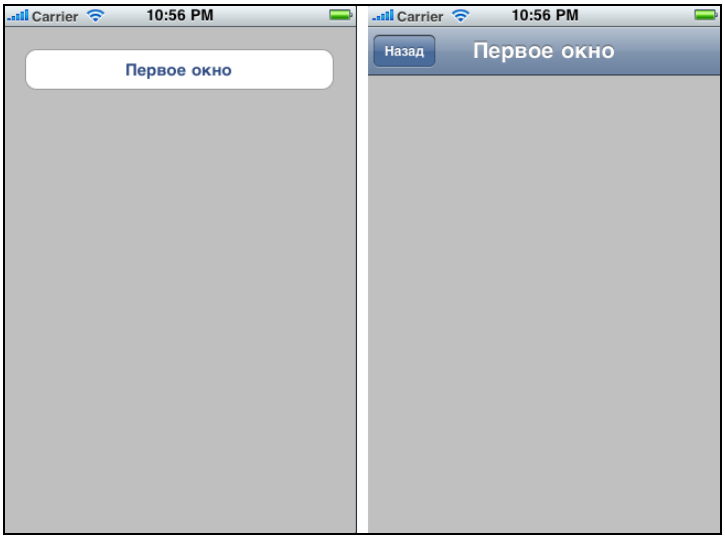


Рис. 4.2. Переключение окон в программе

На этом главу о переключении окон можно было бы завершить. Но как обмениваться данными между окнами? Ведь достаточно редко окно существует само по себе, в большинстве случаев необходимо передать в него какие-либо данные и получить ответные действия пользователя. Для примера дополним созданную программу, добавив в главном окне вывод количества дней относительно даты, введенной пользователем. Анализ полученных от другого окна значений уже рассматривался в предыдущей главе при изучении класса `UIAlertView`, здесь необходимо будет реализовать нечто подобное.

Для того чтобы обеспечить передачу уведомлений от окна к вызвавшему его классу, необходимо сделать три шага:

1. Создать "протокол" (protocol), описывающий необходимые функции.
2. Вызвать из создаваемого (второго) окна требуемую функцию.
3. Добавить в вызывающий класс функцию (или функции), соответствующую созданному протоколу.

Рассмотрим эти шаги подробнее. Добавим в класс `FirstView` возможность ввода даты и описание функции обработки действия пользователя, как в листинге 4.3. Новые строки выделены жирным шрифтом.

Листинг 4.3. Класс окна `FirstView` с поддержкой передачи уведомлений

Файл FirstView.h:

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

@protocol FirstViewDelegate
- (void)onDateChanged:(int)nDays;
@end

@interface FirstView : UIViewController
{
        IBOutlet UIDatePicker *m_cDatePicker;
        id <FirstViewDelegate> delegate;
}

@property (nonatomic, retain) UIDatePicker *m_cDatePicker;
@property (nonatomic, assign) id delegate;

- (IBAction) onButtonBack:(id)sender;
- (IBAction) onButtonContinue:(id)sender;

@end
```

Отметим ключевое слово `@protocol`, описывающее функции в `FirstView`, и объект `delegate` вызова функции. Реализация класса `FirstView` показана в листинге 4.4.

Листинг 4.4. Реализация класса `FirstView` с поддержкой передачи уведомлений

```
#import "FirstView.h"

@implementation FirstView
@synthesize m_cDatePicker;
@synthesize delegate;

// Implement viewDidLoad to do additional setup after loading the view,
// typically from a nib.
- (void)viewDidLoad
{
    [super viewDidLoad];
}

- (IBAction) onButtonBack:(id)sender
{
     [self dismissModalViewControllerAnimated:YES];
```

```
}

- (IBAction) onButtonContinue:(id)sender
{
    NSDate *date = m_cDatePicker.date;
    NSTimeInterval ti = [date timeIntervalSinceNow];
    int days = (int)(ti/(24*60*60));

    [self.delegate onDateChanged: abs(days)];

    [self dismissModalViewControllerAnimated:YES];
}

- (void)didReceiveMemoryWarning
{
    // Releases the view if it doesn't have a superview.
    [super didReceiveMemoryWarning];
}

- (void)viewDidUnload
{
    [super viewDidUnload];
}

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

@end
```

Важно отметить две функции — `onButtonBack` и `onButtonContinue`. Первая функция выполняет "обычное" закрытие окна, без каких-либо дополнительных действий. Это происходит при нажатии кнопки **Back**. Вторая функция вызывается по нажатию кнопки **Continue**, и в ней помимо закрытия окна осуществляется подсчет диапазона дат и вызов функции `onDateChanged`, описанной в секции `@protocol`. Для подсчета интервала дат используется встроенная функция класса `NSDate timeIntervalSinceNow`, возвращающая интервал в секундах. Необходимо разделить полученное число на количество секунд в одном дне, что и производится в приведенном коде.

Где же находится функция `onDateChanged`, вызов которой осуществляется в `onButtonContinue`? Ведь сам протокол содержит лишь описание функции, но не ее реализацию. Действительно, для корректного функционирования необходимо дописать функцию в вызывающий класс и инициализировать объект `delegate`, как показано в листинге 4.5.

Листинг 4.5. Активация окна `FirstView` с использованием объекта `delegate`

Файл TestUIViewsViewController.h:

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

@interface TestUIViewsViewController :
                              UIViewController<FirstViewDelegate>
{
    IBOutlet UILabel *m_cDateLabel;
}

@property (nonatomic, retain) UILabel *m_cDateLabel;

- (IBAction) onShowFirst:(id)sender;

@end
```

Файл TestUIViewsViewController.m (фрагмент):

```
- (IBAction) onShowFirst:(id)sender
{
     FirstView *wnd1 =  [[FirstView alloc] initWithNibName:@"FirstView"
                          bundle:nil];
     wnd1.delegate = self;
     [self presentModalViewController:wnd1 animated:YES];
     [wnd1 release];
}

- (void)onDateChanged:(int)nDays
{
    m_cDateLabel.text = [NSString stringWithFormat:@"%d days", nDays];
}
```

Теперь все встает на свои места. Мы инициализировали объект `wnd1.delegate` указателем на текущий класс (`self`) с функцией `onDateChanhed`, описанной в протоколе. По сути, механизм создания протокола функционирует через вызов callback-функции по указателю.

Теперь можно тестировать созданную программу. Если все было сделано правильно, интерфейс приложения должен выглядеть, как на рис. 4.3.

Кнопкой **Диапазон дат** должно открываться окно ввода даты. По нажатию кнопки **Продолжить** окно закроется, а в главном окне появится количество дней.

Очевидно, что количество функций в секции `@protocol` может быть более одной, что позволяет обрабатывать различные варианты действий пользователя.

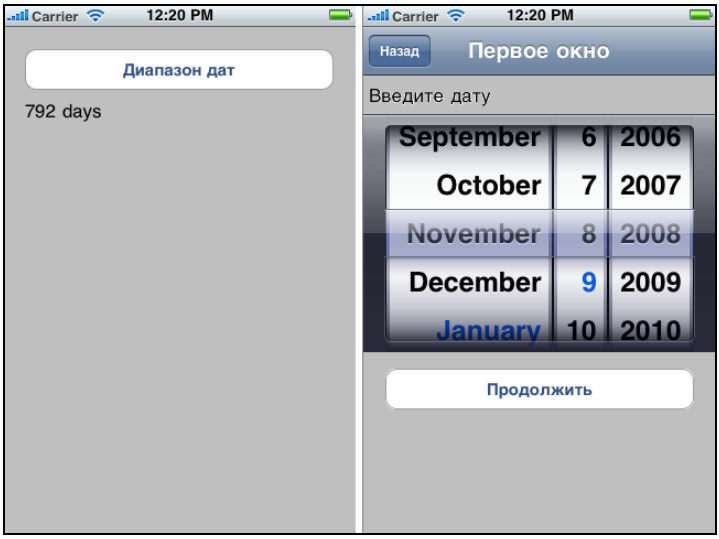


Рис. 4.3. Переключение окон в программе с возвратом значения

Таблицы с настраиваемыми ячейками

В предыдущей главе рассматривался простой способ табличного представления данных. Однако в большинстве программ подобного текстового вывода недостаточно. Рассмотрим способ более сложного вывода данных произвольной формы.

Обратимся еще раз к загрузке и инициализации табличных данных, как показано в листинге 4.6.

Листинг 4.6. Вывод данных в виде таблицы (фрагмент кода)

```
-(NSInteger)numberOfSectionsInTableView:(UITableView *)tableView
{
    return 1;
}

-(NSInteger)tableView:(UITableView *)tableView
                                          numberOfRowsInSec-
tion:(NSInteger)section
{
    return 8;
}

- (UITableViewCell *)tableView:(UITableView *)tableView
                        cellForRowAtIndexPath:(NSIndexPath *)indexPath
```

```
{

    static NSString *MyIdentifier = @"IDDQD"; // any ID

    UITableViewCell *cell = [m_pItemsTable
                          dequeueReusableCellWithIdentifier:MyIdentifier];

    // If no cell is available, create a new one using
    // the given identifier.
    if (cell == nil)
    {
        // Use the default cell style.
        cell = [[[UITableViewCell alloc]
                           initWithStyle:UITableViewCellStyleDefault
                           reuseIdentifier:MyIdentifier] autorelease];
    }

    NSArray *temp = [[NSArray alloc] initWithObjects:@"Меркурий",
                         @"Венера", @"Земля", @"Марс",
                         @"Юпитер", @"Сатурн", @"Уран", @"Нептун", nil];

    // Set text
    cell.textLabel.text = [temp objectAtIndex:indexPath.row];

    [temp release];

     return cell;
}

- (NSIndexPath *)tableView:(UITableView *)tableView
                       willSelectRowAtIndexPath:(NSIndexPath *)indexPath
{
     return indexPath;
}
```

В настоящее же время нас будет интересовать функция `cellForRowAtIndexPath`. Возвращая ячейку (`cell`) для определенного индекса, она позволяет значительно расширить функциональность `UITableView`, выводя в него произвольные данные.

Расширим функциональность предыдущей программы: будем выводить в таблице не только названия планет, но и их фотографии. Создадим новый проект TestUITable2, добавим в основной класс компонент `UITableView`. Откроем XIB-файл в Interface Builder, добавим в него элемент **Table View** и установим связи, как показано на рис. 4.4.

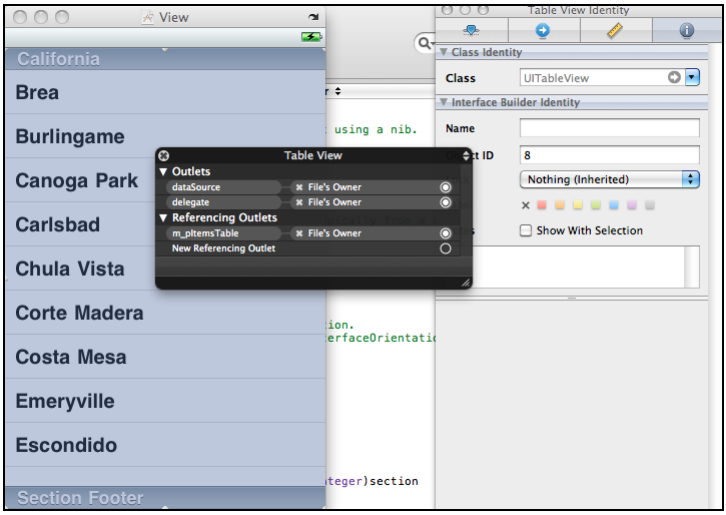


Рис. 4.4. Настройка `UITableView`

Если этого не сделать, то соответствующие функции не будут получать уведомления от системы, и таблица не будет отображаться.

Для создания ячейки пользователя нужно выполнить несколько действий:

1. Добавить в проект класс, наследуемый от `UITableViewCell`, как показано на рис. 4.5.

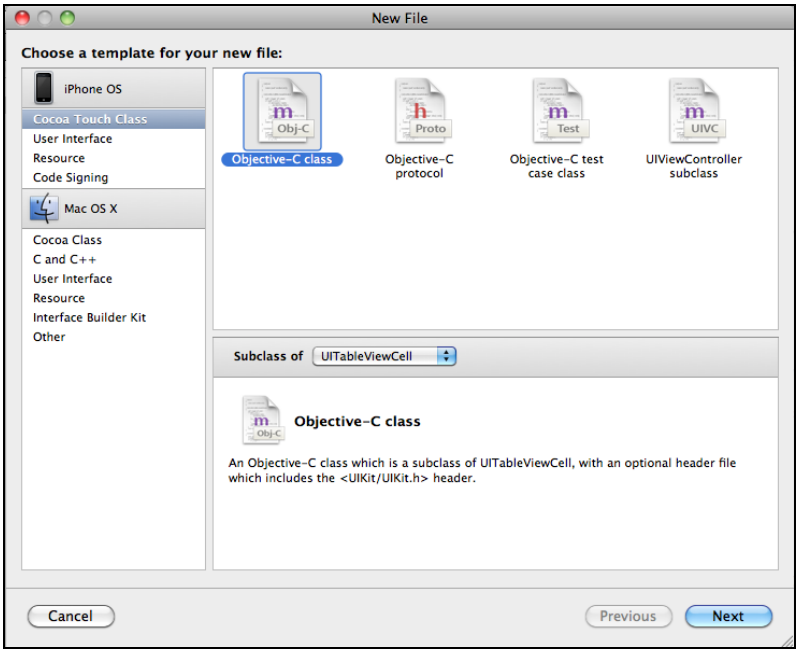


Рис. 4.5. Добавление класса `TableViewCell`

2. Добавить в проект новый XIB-файл и вставить в него элемент Table Cell.
3. Добавить в него требуемые элементы интерфейса.
4. Добавить соответствующие переменные в созданный класс и установить связи.

Выполним эти шаги на практике. Добавим в проект класс, наследуемый от `UITableViewCell`, назовем его, по аналогии с остальными файлами проекта, `TestUITable2ViewCell`. XCode создаст два файла: TestUITable2ViewCell.h и TestUITable2ViewCell.m. Затем добавим в проект новый XIB-файл с названием TestUITable2ViewCell.xib, в котором поместим элемент **Table Cell.**

Поскольку нам нужно выводить названия планет, разместим в ячейке два элемента: текстовое поле (`UILabel`) для названия планеты и изображение (`UIImageView`) для фотографии. Внешний вид созданной ячейки и XIB-файла должен соответствовать рис. 4.6. В поле `Identifier` впишем идентификатор ячейки, назовем его `tblCellView`, как показано на этом же рисунке. Это поле понадобится в дальнейшем.

Следующим шагом необходимо добавить в созданный нами класс `TestUITable2ViewCell` переменные, соответствующие элементам интерфейса. Создадим два свойства `property` для `UILabel` и `UIImageView`, как показано в листинге 4.7.

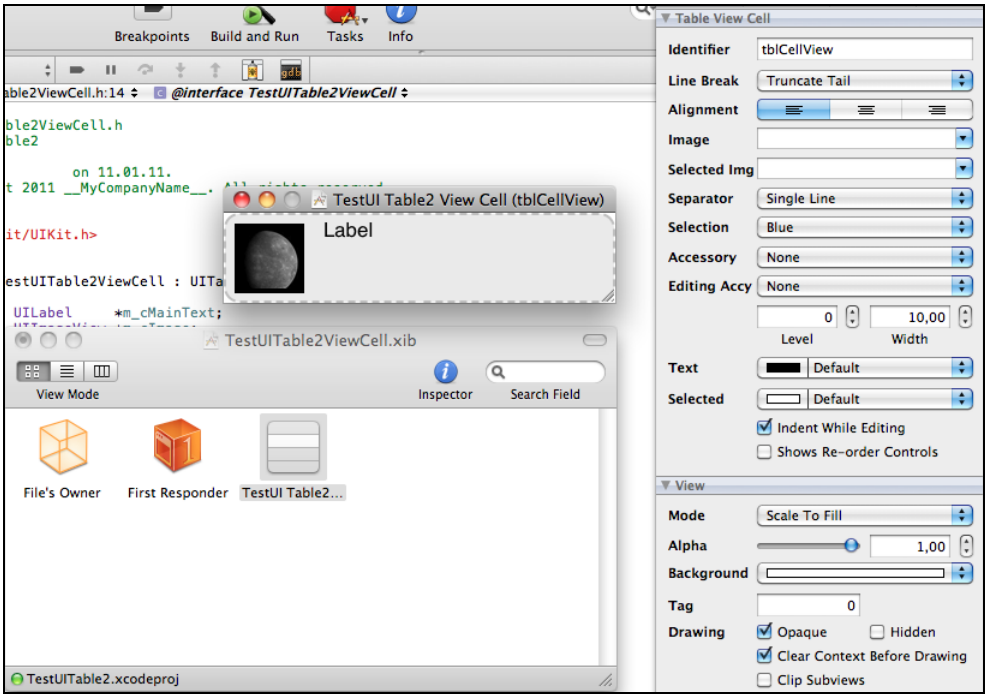


Рис. 4.6. Ячейка Table Cell

Листинг 4.7. Класс, наследуемый от `UITableViewCell`

Файл TestUITable2ViewCell.h:

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

@interface TestUITable2ViewCell : UITableViewCell
{
       IBOutlet UILabel       *m_cMainText;
       IBOutlet UIImageView       *m_cImage;
}

@property(retain, nonatomic) UILabel           *m_cMainText;
@property(retain, nonatomic) UIImageView *m_cImage;


@end
```

Файл TestUITable2ViewCell.m:

```
#import "TestUITable2ViewCell.h"

@implementation TestUITable2ViewCell
@synthesize m_cMainText;
@synthesize m_cImage;

- (id)initWithStyle:(UITableViewCellStyle)style
                        reuseIdentifier:(NSString *)reuseIdentifier
{
    if ((self = [super initWithStyle:style
                                      reuseIdentifier:reuseIdentifier]))
    {
    }
    return self;
}

- (void)setSelected:(BOOL)selected animated:(BOOL)animated
{

    [super setSelected:selected animated:animated];
}

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

@end
```

Код практически не отличается от использования этих же компонентов в "обычных" диалоговых окнах, за исключением новых функций `initWithStyle` и `setSelected`, которые мы оставляем без изменений. Свойства `property` создаются и удаляются таким же образом, что и для всех остальных окон.

После того как класс создан, можно установить связи между кодом и элементами интерфейса, как показано на рис. 4.7.

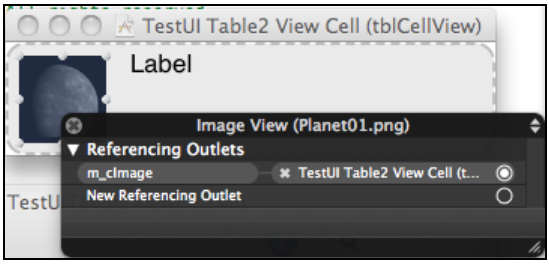


Рис. 4.7. Установление связи в ячейке

В данном примере необходимо установить две связи, с элементом **Label** и изображением (вторая связь на рисунке не показана). Теперь класс `TestUITable2ViewCell` может быть использован в программе.

И наконец, необходимо изменить функцию `cellForRowAtIndexPath`, чтобы она возвращала созданную нами ячейку (листинг 4.8).

Листинг 4.8. Загрузка `TableViewCell` в таблицу

```
- (UITableViewCell *)tableView:(UITableView *)tableView
                         cellForRowAtIndexPath:(NSIndexPath *)indexPath
{
     TestUITable2ViewCell *cell = (TestUITable2ViewCell *)[tableView
                    dequeueReusableCellWithIdentifier:@"tblCellView"];
     if (cell == nil)
     {
          cell = [[[NSBundle mainBundle]
                    loadNibNamed:@"TestUITable2ViewCell" owner:self
                    options:nil] lastObject];
     }

     return cell;
}
```

Функция пытается загрузить элемент с идентификатором `tblCellView`, и если такой элемент не найден, он загружается из XIB-файла.

Теперь можно скомпилировать и запустить программу. Однако все элементы ячеек будут пустыми, ведь для каждой ячейки необходимо указать текст и изобра-

жение. Текст мы уже задавали, как можно видеть в листинге 4.6, а изображения еще нет. Добавим в проект восемь изображений планет и назовем соответствующие файлы от Planet01.png до Planet08.png. Теперь в листинге 4.9 мы можем привести окончательный код для работы с таблицей.

Листинг 4.9. Окончательный вариант `TestUITable2ViewController`

```
//  TestUITable2ViewController.m
//  TestUITable2
//

#import "TestUITable2ViewController.h"
#import "TestUITable2ViewCell.h"

@implementation TestUITableViewController
@synthesize m_pItemsTable;

-(NSInteger)numberOfSectionsInTableView:(UITableView *)tableView
{
    return 1;
}

-(NSInteger)tableView:(UITableView *)tableView
                                          numberOfRowsInSec-
tion:(NSInteger)section
{
    return 8;
}

- (UITableViewCell *)tableView:(UITableView *)tableView
                        cellForRowAtIndexPath:(NSIndexPath *)indexPath
{
     TestUITable2ViewCell *cell = (TestUITable2ViewCell *)[tableView
                    dequeueReusableCellWithIdentifier:@"tblCellView"];
     if (cell == nil)
     {
          cell = [[[NSBundle mainBundle]
                    loadNibNamed:@"TestUITable2ViewCell" owner:self
                    options:nil] lastObject];
     }

     int index = indexPath.row;
     NSArray *temp = [[NSArray alloc] initWithObjects:@"Меркурий",
                          @"Венера", @"Земля", @"Марс",
                          @"Юпитер", @"Сатурн", @"Уран", @"Нептун", nil];
```

```
    // Set text and image
    cell.m_cMainText.text = [temp objectAtIndex:index];
    cell.m_cImage.image = [UIImage imageNamed:
                [NSString stringWithFormat:@"Planet0%d.png", index+1]];

    // Set color
    UIColor *color = index & 1?[UIColor lightGrayColor]:
                               [UIColor whiteColor];
    cell.m_cMainText.backgroundColor = color;
    cell.m_cImage.backgroundColor = color;
    cell.contentView.backgroundColor = color;

    [temp release];

    return cell;
}

- (NSIndexPath *)tableView:(UITableView *)tableView
                      willSelectRowAtIndexPath:(NSIndexPath *)indexPath
{
     return indexPath;
}

- (void)tableView:(UITableView*)tableView
                      didSelectRowAtIndexPath:(NSIndexPath *)indexPath
{
     UIAlertView *simpleAlert = [[UIAlertView alloc]

                      initWithTitle:@"Объект был выбран"
                      message:@"..."
                      delegate:self
                      cancelButtonTitle:@"OK"
                      otherButtonTitles:nil, nil];
      [simpleAlert show];
      [simpleAlert release];
}

- (void)didReceiveMemoryWarning
{
     // Releases the view if it doesn't have a superview.
    [super didReceiveMemoryWarning];
}

- (void)viewDidUnload
```

```
{
}

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

@end
```

Мы воспользовались созданными свойствами `m_cMainText` и `m_cImage` для внесения текста и изображений. Для загрузки графических файлов мы воспользовались уже описанной ранее функцией `imageNamed` класса `UIImage`. Для более удобного чтения фоновый цвет таблицы задан в свойстве `backgroundColor`. У нечетных чисел младший бит всегда равен единице, так что чередование цветов задается в следующем коде:

```
UIColor *color = index & 1?[UIColor lightGrayColor]:
                           [UIColor whiteColor];
```

Используется тернарная операция `?` (имеющаяся в языке C), что допускается в Objective-C.

Теперь можно скомпилировать и выполнить окончательную версию программы, результат должен быть примерно похож на рис. 4.8.

Как и в предыдущем случае, стоит отметить, что размещение данных прямо в тексте программы не является правильным с точки зрения идеологии программирования, но в данном примере используется для наглядности и минимизации объема кода. В реальных проектах надо стараться разделять данные и код, обслуживающий графический интерфейс.

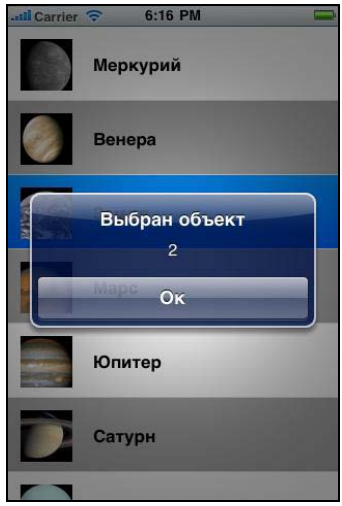


Рис. 4.8. Окончательный вариант таблицы

Таблицы с подразделами

Следующей особенностью класса `UITableView` является возможность вывода таблицы, состоящей из нескольких подразделов. Это позволяет группировать данные логически, отделяя одну часть от другой. Дополним программу, показывающую список планет солнечной системы: помимо "обычных" планет, будем выводить их спутники, которые назовем "малыми планетами".

Для создания программы необходимо выполнить несколько шагов.

1. Создать новое приложение, поместить на главном окне элемент `UITableView`.
2. Добавить соответствующую переменную и установить связи через Interface Builder.
3. Добавить функции, обеспечивающие наполнение таблицы.

Первые два пункта уже рассматривались ранее, на последнем стоит остановиться более подробно. Мы уже использовали протокол `UITableViewDataSource`, функции которого необходимо реализовать в программе. Для вывода подразделов нужно будет изменить следующие функции:

- `numberOfSectionsInTableView` — количество секций;
- `numberOfRowsInSection`, возвращающую количество элементов в каждой секции;
- `titleForHeaderInSection` — заголовок секции;
- `cellForRowAtIndexPath` — заполнение ячейки текстом.

Нам необходимо вывести названия больших и малых планет Солнечной системы (список планет гораздо больше, но мы не ставим целью создать полный астрономический каталог).

Листинг 4.10. Вывод таблицы с подразделами

Файл TestUITable3ViewController.h:

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

@interface TestUITable3ViewController : UIViewController <UITableViewDelegate,

UITableViewDataSource>
{
       IBOutlet UITableView  *m_pItemsTable;
}

@property(retain, nonatomic) UITableView   *m_pItemsTable;

@end
```

Файл TestUITable3ViewController.m:

```
//
//  TestUITable3ViewController.m
//  TestUITable3
//

#import "TestUITable3ViewController.h"
```

```
@implementation TestUITable3ViewController
@synthesize m_pItemsTable;

// Implement viewDidLoad to do additional setup after loading the view,
// typically from a nib.
- (void)viewDidLoad
{
    [super viewDidLoad];
}

-(NSInteger)numberOfSectionsInTableView:(UITableView *)tableView
{
    return 2;
}

-(NSInteger)tableView:(UITableView *)tableView
                        numberOfRowsInSection:(NSInteger)section
{
    if (section == 0)
        return 8;
    if (section == 1)
        return 7;

    return 0;
}

- (NSString *)tableView:(UITableView *)tableView
                      titleForHeaderInSection:(NSInteger)section
{
    if(section == 0)
        return @"Планеты";
    else
        return @"Малые планеты";
}

- (UITableViewCell *)tableView:(UITableView *)tableView
                     cellForRowAtIndexPath:(NSIndexPath *)indexPath
{
    static NSString *MyIdentifier = @"IDDQD"; // any ID

    UITableViewCell *cell = [m_pItemsTable
                  dequeueReusableCellWithIdentifier:MyIdentifier];

    // If no cell is available, create a new one using
    // the given identifier.
    if (cell == nil)
    {
        // Use the default cell style.
```

```
        cell = [[[UITableViewCell alloc]
                        initWithStyle:UITableViewCellStyleDefault
                        reuseIdentifier:MyIdentifier] autorelease];
    }

    const int row = indexPath.row, section = indexPath.section;
    if (section == 0)
    {
        NSArray *temp = [[NSArray alloc] initWithObjects: @"Меркурий",
                         @"Венера", @"Земля", @"Марс", @"Юпитер",
                         @"Сатурн", @"Уран", @"Нептун", nil];
        cell.textLabel.text = [temp objectAtIndex:row];
        [temp release];
    } else
    {
        NSArray *temp = [[NSArray alloc] initWithObjects: @"Луна",
                        @"Фобос", @"Деймос", @"Ио", @"Европа", @"Ганимед",
                        @"Каллисто", nil];
        cell.textLabel.text = [temp objectAtIndex:row];
        [temp release];
    }

    return cell;
}

- (NSIndexPath *)tableView:(UITableView *)tableView
                        willSelectRowAtIndexPath:(NSIndexPath *)indexPath
{
    return indexPath;
}

- (void)tableView:(UITableView *)tableView
                        didSelectRowAtIndexPath:(NSIndexPath *)indexPath
{
     NSString *s_info = [NSString stringWithFormat:@"%d.%d",
                                  indexPath.section, indexPath.row];

     UIAlertView *simpleAlert = [[UIAlertView alloc]
                           initWithTitle:@"Выбран объект"
                           message:s_info
                           delegate:self
                           cancelButtonTitle:@"Ок"
                           otherButtonTitles:nil,nil];
     [simpleAlert show];
     [simpleAlert release];
}
```

```
- (void)didReceiveMemoryWarning
{
    // Releases the view if it doesn't have a superview.
    [super didReceiveMemoryWarning];
}

- (void)viewDidUnload
{
}

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

@end
```

Помимо названий планет, в листинге 4.10 можно видеть заголовки секций, которые задаются в функции `titleForHeaderInSection`. Функция `numberOfRowsInSection` возвращает число 8 в секции планет и 7 — малых планет. При необходимости придется не только изменить массив, но и не забыть изменить возвращаемое этой функцией значение. Это не очень удобно, но, как и в предыдущих примерах, такой подход используется для максимального упрощения кода. В реальном проекте было бы более эффективным хранить массив с данными как член класса, тогда количество элементов можно было бы подсчитывать автоматически.

Помимо показанного на рис. 4.9 режима вывода, `UITableView` поддерживает еще один режим, в котором таблица разбивается на части.

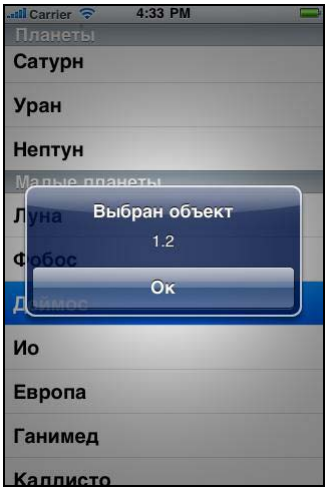


Рис. 4.9. Вид таблицы с разделами

Для включения этого режима следует установить свойство **Style** (стиль) таблицы в **Grouped**, как показано на рис. 4.10.

Результат показан на рис. 4.11. Интересно отметить, что помимо функции `titleForHeaderInSection`, задающей название заголовка, существует еще функция `titleForFooterInSection`, с помощью которой можно задавать пояснения в нижней части таблицы.

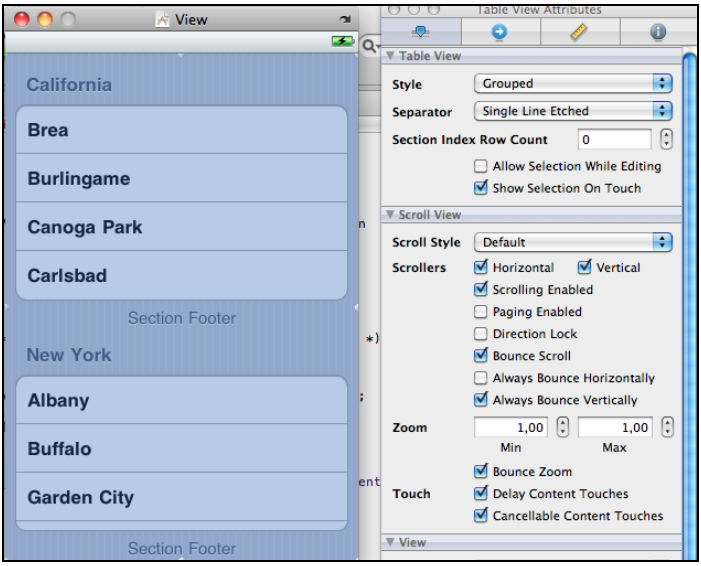


Рис. 4.10. Настройка `UITableView`

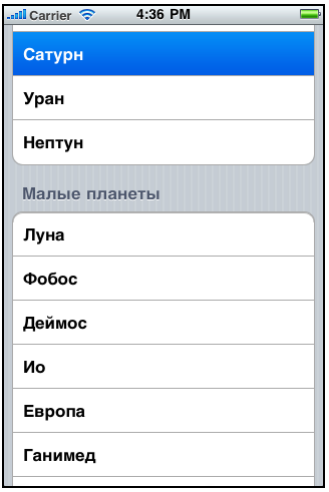


Рис. 4.11. Вид таблицы в режиме Grouped

Можно также задать пользовательский компонент для заголовка и подписи с помощью переопределения функций `viewForHeaderInSection` и `viewForFooterInSection`.

На этом мы закончим с таблицами и рассмотрим другой способ вывода данных сложной формы.

Вывод информации в формате HTML

Представление данных с помощью HTML дает большие возможности, ведь способы форматирования данных в этом языке весьма велики. Это разнообразные шрифты, таблицы, изображения, указание цвета с помощью CSS-стилей и многое другое. Реализуем вывод названий планет и их изображений с помощью HTML:

1. Создадим новый проект под названием TestUIWebView, в главном окне поместим компонент `UIWebView`.
2. Добавим в класс `TestUIWebViewViewController` компонент типа `UIWebView` и укажем, что он является свойством (`property`) класса.
3. Установим связи с помощью Interface Builder.
4. В функции инициализации окна загрузим нужные данные в формате HTML.

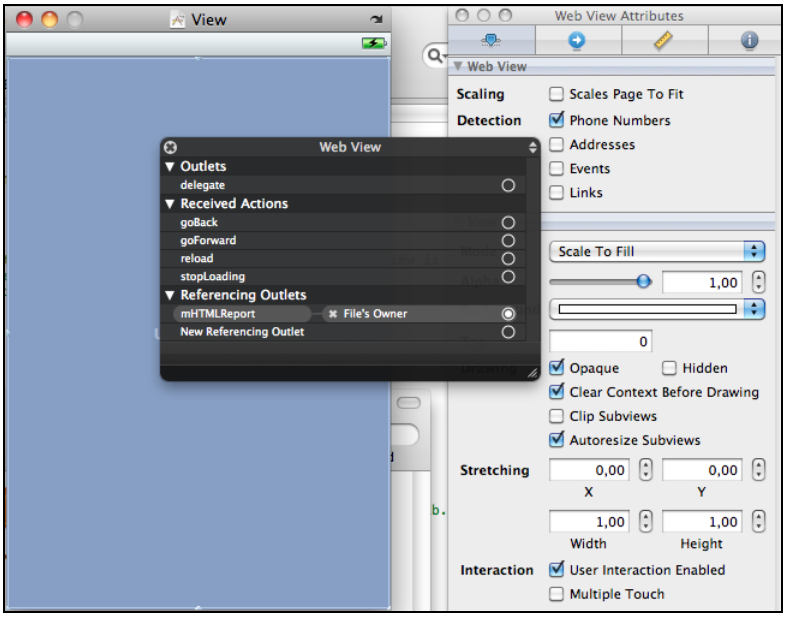


Рис. 4.12. Свойства компонента `UIWebView`

Первые три пункта в особых комментариях не нуждаются. После создания компонента и установления связей вид `UIWebView` должен быть, как на рис. 4.12.

Сам компонент растянут на все окно, поэтому сливается с ним. Чтобы сделать программу более интересной, разместим на диалоговом окне две кнопки: одна будет показывать список основных планет, другая — малые планеты.

Начнем с инициализации компонента `UIWebView` в листинге 4.11.

Листинг 4.11. Инициализация `UIWebView`

Файл TestUIWebViewViewController.h:

```
//  TestUIWebViewViewController.h
//  TestUIWebView
//

#import <UIKit/UIKit.h>

@interface TestUIWebViewViewController : UIViewController
{
    IBOutlet UIWebView   *mHTMLReport;
}

- (IBAction) onShowPlanets:(id)sender;
- (IBAction) onShowPlanetsSmall:(id)sender;
```

```
@property(retain, nonatomic) UIWebView   *mHTMLReport;

@end
```

Файл TestUIWebViewViewController.m:

```
//  TestUIWebViewViewController.m
//  TestUIWebView
//

#import "TestUIWebViewViewController.h"

@implementation TestUIWebViewViewController
@synthesize mHTMLReport;


// Implement viewDidLoad to do additional setup after loading the view,
// typically from a nib.
- (void)viewDidLoad
{
    [super viewDidLoad];

     NSString *str = @"Нажмите любую кнопку";
     [mHTMLReport loadHTMLString:str baseURL:[NSURL URLWithString:@""]];
}

- (IBAction) onShowPlanets:(id)sender
{
}

- (IBAction) onShowPlanetsSmall:(id)sender
{
}

- (void)didReceiveMemoryWarning
{
    // Releases the view if it doesn't have a superview.
    [super didReceiveMemoryWarning];
}

- (void)viewDidUnload
{
    // Release any retained subviews of the main view.
    // e.g. self.myOutlet = nil;
}

- (void)dealloc
{
```

```
    [mHTMLReport release];
    [super dealloc];
}

@end
```

Как можно видеть, загрузка HTML-строки сводится к вызову функции `loadHTMLString` объекта `mHTMLReport` — тут все действительно просто. Однако чтобы загрузить картинку в HTML, нужно знать ее локальный путь. Для этого используется код, показанный в листинге 4.12.

Листинг 4.12. Загрузка изображений в HTML

```
NSString *path  = [[NSBundle mainBundle] resourcePath];
path = [path stringByReplacingOccurrencesOfString:@"/" withString:@"//"];
path = [path stringByReplacingOccurrencesOfString:@" "
                                          withString:@"%20"];

NSString *html = @"Пример изображения<BR/><img src=\"file.jpg\"/>";
[mHTMLReport loadHTMLString:html
            baseURL:[NSURL URLWithString:
            [NSString stringWithFormat:@"file:/%@//", path]]];
```

Основной хитростью является использование параметра `baseURL` пути к файлам программы. Для того чтобы узнать путь, используется функция `resourcePath`, значение которой преобразуется в строку, соответствующую стандарту HTML. Конечно, можно было бы просто сделать таблицу планет в любом HTML-редакторе и вставить в программу, но интереснее сформировать HTML программно, как это могло бы быть в реальном проекте (листинг 4.13).

Листинг 4.13. Таблица для вывода в HTML

```
<table width=\"100%\" border=\"0\">
  <tr>
    <td width=\"60\"><div align=\"center\">
                      <img src=\"photo.png\"/></div>
    </td>
    <td><h2>text</h2></td>
  </tr>
   …
  <tr>
    <td width=\"60\"><div align=\"center\">
                      <img src=\"photo.png\"/></div>
    </td>
    <td><h2>text</h2></td>
  </tr>
</table>
```

Зная это, несложно написать функцию программного формирования таблицы, как показано в листинге 4.14.

Листинг 4.14. Код программы для вывода таблицы в HTML

```
NSArray *names = [[NSArray alloc] initWithObjects: @"Меркурий",
                      @"Венера", @"Земля", @"Марс", @"Юпитер",
                      @"Сатурн", @"Уран", @"Нептун", nil];

NSString *html = @"<body>"
                 @"<h1>Планеты солнечной системы</h1>"
                 @"<table width=\"100%\" border=\"0\">";
for(int p=0; p<[names count]; p++)
{
     NSString *h1 = [NSString stringWithFormat:@"<td width=\"60\">"
             @"<div align=\"center\"><img src=\"Planet0%d.png\"/></div>"
             @"</td>", p+1],
              *h2 = [NSString stringWithFormat:@"<td>"
             @"<h2>%@</h2>"
             @"</td>", [names objectAtIndex:p]];

     html = [NSString stringWithFormat:@"%@ <tr>%@ %@</tr>",
                                       html, h1, h2];
}

html = [NSString stringWithFormat:@"%@ "
                                  @"</table>"
                                  @"</body></html>", html];

NSString *imagePath = [[NSBundle mainBundle] resourcePath];
imagePath = [imagePath stringByReplacingOccurrencesOfString:@"/"
                                        withString:@"//"];
imagePath = [imagePath stringByReplacingOccurrencesOfString:@" "
                                        withString:@"%20"];

[mHTMLReport loadHTMLString:html baseURL:[NSURL URLWithString:
                 [NSString stringWithFormat:@"file:/%@//",imagePath]]];
[names release];
```

Для формирования строки используется уже известная функция `stringWithFormat`. В отличие от C++, `NSString` не позволяет добавлять текст с помощью переопределенного оператора `+=`, поэтому для добавления текста в конец строки используется следующая конструкция:

```
NSString *str = @"Текст";
str = [NSString stringWithFormat:@"%@ Новый текст", str];
```

Как и в предыдущем примере с `UItableView`, изображения, выводимые с помощью HTML, должны храниться в ресурсах программы. В iOS все необходимые файлы просто хранятся в одной папке с программой. Надо заметить, что пользователь устройства доступа к этой папке не имеет, поэтому не может удалить или испортить файлы программы, как говорилось еще *в главе 1*.

Теперь мы можем вставить приведенный код в функцию `onShowPlanets` и посмотреть на рис. 4.13.

Рис. 4.13. Вывод таблицы с помощью `UIWebView`

Можно еще чередовать цвета, как было показано в прошлом разделе.

Следующим шагом является вывод списка малых планет. Можно скопировать функцию в `onShowPlanetsSmall` и поменять лишь текстовый массив, но два больших и почти повторяющихся фрагмента кода выглядят некрасиво. Поэтому сделаем код формирования таблицы отдельной функцией, которая на вход будет принимать три параметра: заголовок таблицы, массив названий планет и массив изображений.

Окончательный вариант функции вывода показан в листинге 4.15.

Листинг 4.15. Функция вывода таблицы в HTML

```
- (void) ShowPlanetsList: (NSString*)sHeader: (NSArray*)pPlanets: (NSAr-
ray*)pImages
{
    NSString *imagePath = [[NSBundle mainBundle] resourcePath];
    imagePath = [imagePath stringByReplacingOccurrencesOfString:@"/"
                                            withString:@"//"];
    imagePath = [imagePath stringByReplacingOccurrencesOfString:@" "
                                            withString:@"%20"];
```

```
    NSString *html = [NSString stringWithFormat: @"<body>"

                @"<h1>%@</h1>"
                @"<table width=\"100%\" border=\"0\">", sHeader];

    for(int p=0; p<[pPlanets count]; p++)
    {
        NSString *h1 = [NSString stringWithFormat:@"<td width=\"60\">"
                                  @"<div align=\"center\">"
                                  @"<img src=\"%@\"/></div></td>",
                                           [pImages objectAtIndex:p]],
                *h2 = [NSString
                           stringWithFormat:@"<td><h2>%@</h2></td>",
                            [pPlanets objectAtIndex:p]];

        html = [NSString stringWithFormat:@"%@ <tr>%@ %@</tr>",
                                         html, h1, h2];
    }

    html = [NSString stringWithFormat:@"%@"
                                      @"</table>"
                                      @"</body></html>", html];

    [mHTMLReport loadHTMLString:html baseURL:[NSURL URLWithString:
              [NSString stringWithFormat:@"file:/%@//",imagePath]]];
}
```

Мы можем переписать функции-обработчики нажатия кнопок, как показано в листинге 4.16, сведя дублирование кода к минимуму.

Листинг 4.16. Вывод таблиц данных

```
- (IBAction) onShowPlanets:(id)sender
{
       NSArray *names = [[NSArray alloc] initWithObjects: @"Меркурий",
                                  @"Венера", @"Земля", @"Марс",
                                  @"Юпитер", @"Сатурн", @"Уран",
                                  @"Нептун", nil];
       NSArray *images= [[NSArray alloc] initWithObjects:@"Planet01.png",
                      @"Planet02.png", @"Planet03.png", @"Planet04.png",
                      @"Planet05.png", @"Planet06.png", @"Planet07.png",
                      @"Planet08.png", nil];

       [self ShowPlanetsList: @"Планеты солнечной системы": names:
                                                            images];
```

```
        [names release];
        [images release];
}

- (IBAction) onShowPlanetsSmall:(id)sender
{
        NSArray *names = [[NSArray alloc] initWithObjects: @"Луна",
                          @"Фобос", @"Деймос", @"Ио",
                          @"Европа", @"Ганимед", @"Каллисто", nil];
        NSArray *images= [[NSArray alloc] initWithObjects:@"PlanetNA.png",
                          @"PlanetNA.png", @"-", @"-",
                          @"-", @"-", @"-", nil];

        [self ShowPlanetsList: @"Малые планеты Солнечной системы":names:
                                                                images];

        [names release];
        [images release];
}
```

На рис. 4.14 отсутствующие изображения заменяются знаком вопроса, что выглядит вполне логично: не так просто вывести данные со сложным форматированием и изображениями.

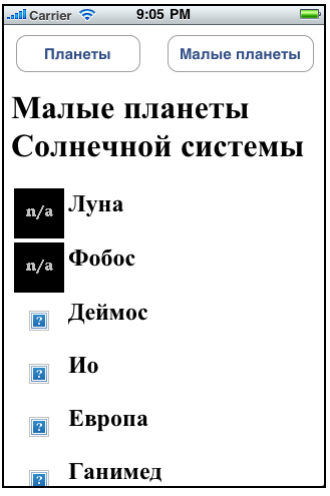


Рис. 4.14. Отображение второй таблицы

Добавим в конец списка ссылку **Показать все** — общий список больших и малых планет. Очевидно, что для этого нужно обрабатывать сообщения от компонента `UIWebView`, и такая возможность есть — знакомый нам принцип делегатов, опи-

санный в разделе "UIWebViewDelegate Protocol Reference" документации. Нам нужна функция `shouldStartLoadWithRequest`, которая вызывается до начала загрузки документа.

Интерактивность данных обеспечивается следующим набором действий:

1. Указать с помощью Interface Builder, что наш класс окна является получателем данных от `UIWebView`.
2. Добавить в код формирования HTML нужные ссылки.
3. Добавить в класс окна функцию `shouldStartLoadWithRequest` и вставить в нее обработку кода соответствующих ссылок.

Что касается первого пункта, то он понятен, но для разнообразия мы пойдем другим путем. Помимо Interface Builder нужные связи можно установить программно, для этого в функцию `viewDidLoad` нужно добавить следующий код (листинг 4.17).

Листинг 4.17. Доработка функции `viewDidLoad`

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

    mHTMLReport.delegate = self;
}
```

После этого все уведомления будут приходить к нашему классу.

Следующим шагом мы добавим ссылку к коду формирования HTML (листинг 4.18). Изменим последнюю строку функции `ShowPlanetsList`.

Листинг 4.18. Добавление ссылки к HTML

```
html = [NSString stringWithFormat:@"%@"
                                  @"</table>"
                                  @"<A HREF=\"#ShowAll\">Показать все</A>"
                                  @"</body></html>", html];
```

Мы добавили ссылку `#ShowAll` с помощью атрибута `HREF`. Следующим шагом нужно дополнить функцию `shouldStartLoadWithRequest`, вставив в нее код обработки новой ссылки в листинге 4.19.

Листинг 4.19. Функция `shouldStartLoadWithRequest`

```
- (BOOL)webView:(UIWebView*)webView
       shouldStartLoadWithRequest:(NSURLRequest*)request
       navigationType:(UIWebViewNavigationType)navigationType
{
    if (navigationType == UIWebViewNavigationTypeLinkClicked)
```

```
    {
        NSURL *URL = [request URL];
        NSString *link = [URL fragment];

        if ([link isEqualToString:@"ShowAll"])
        {
            [self onShowAll];
            return NO;
        }
    }
    return YES;
}
```

Мы сначала получаем URL, анализируем его текст с помощью функции `fragment`, и если он соответствует нашей ссылке, то выполняем нужное действие.

Теперь можно соединить все в листинге 4.20.

Листинг 4.20. Вывод данных с помощью `UIWebView`

Файл TestUIWebViewViewController.h:

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

@interface TestUIWebViewViewController :
                                    UIViewController<UIWebViewDelegate>
{
       IBOutlet UIWebView   *mHTMLReport;
}

- (IBAction) onShowPlanets:(id)sender;
- (IBAction) onShowPlanetsSmall:(id)sender;
- (void) onShowAll;
- (void) ShowPlanetsList: (NSString*)sHeader: (NSArray*)pPlanets:
                                              (NSArray*)pImages;

@property(retain, nonatomic) UIWebView   *mHTMLReport;

@end
```

Файл TestUIWebViewViewController.m:

```
//  TestUIWebViewViewController.m
//  TestUIWebView
//

#import "TestUIWebViewViewController.h"

@implementation TestUIWebViewViewController
@synthesize mHTMLReport;
```

```
// Implement viewDidLoad to do additional setup after loading the view,
// typically from a nib.
- (void)viewDidLoad
{
    [super viewDidLoad];

    mHTMLReport.delegate = self;

    NSString *str = @"Нажмите любую кнопку";
    [mHTMLReport loadHTMLString:str baseURL:[NSURL URLWithString:@""]];
}

- (void) ShowPlanetsList: (NSString*)sHeader: (NSArray*)pPlanets:
                                              (NSArray*)pImages
{
    NSString *imagePath = [[NSBundle mainBundle] resourcePath];
    imagePath = [imagePath stringByReplacingOccurrencesOfString:@"/"
                                            withString:@"//"];
    imagePath = [imagePath stringByReplacingOccurrencesOfString:@" "
                                            withString:@"%20"];

    NSString *html = [NSString stringWithFormat: @"<body>"
                      @"<h1>%@</h1>"
                      @"<table width=\"100%\" border=\"0\">", sHeader];

    for(int p=0; p<[pPlanets count]; p++)
    {
        NSString *h1 = [NSString stringWithFormat:@"<td width=\"60\">"
                                  @"<div align=\"center\">"
                                  @"<img src=\"%@\"/></div></td>",
                                            [pImages objectAtIndex:p]],
                *h2 = [NSString
                              stringWithFormat:@"<td><h2>%@</h2></td>",
                                [pPlanets objectAtIndex:p]];

        html = [NSString stringWithFormat:@"%@ <tr>%@ %@</tr>",
                                                    html, h1, h2];
    }

    html = [NSString stringWithFormat:@"%@"
                      @"</table>"
                      @"<A HREF=\"#ShowAll\">"
                      @"Показать все"
                      @"</A>"
                      @"</body></html>", html];

    [mHTMLReport loadHTMLString:html baseURL:[NSURL URLWithString:
                 [NSString stringWithFormat:@"file:/%@//",imagePath]]]; }
```

```
- (IBAction) onShowPlanets:(id)sender
{
       NSArray *names = [[NSArray alloc] initWithObjects: @"Меркурий",
                     @"Венера", @"Земля", @"Марс",
                     @"Юпитер", @"Сатурн", @"Уран",
                     @"Нептун", nil];
       NSArray *images= [[NSArray alloc] initWithObjects:@"Planet01.png",
                     @"Planet02.png", @"Planet03.png", @"Planet04.png",
                     @"Planet05.png", @"Planet06.png", @"Planet07.png",
                     @"Planet08.png", nil];

       [self ShowPlanetsList: @"Планеты солнечной системы": names:
                                                          images];

       [names release];
       [images release];
}

- (IBAction) onShowPlanetsSmall:(id)sender
{
       NSArray *names = [[NSArray alloc] initWithObjects: @"Луна",
                                 @"Фобос", @"Деймос", @"Ио",
                                 @"Европа", @"Ганимед", @"Каллисто", nil];
       NSArray *images= [[NSArray alloc] initWithObjects:@"PlanetNA.png",
                                 @"PlanetNA.png", @"-", @"-",
                                 @"-", @"-", @"-", nil];

       [self ShowPlanetsList: @"Малые планеты Солнечной системы":names:
                                                              images];

       [names release];
       [images release];
}

- (void) onShowAll
{
       NSArray *names= [[NSArray alloc] initWithObjects: @"Меркурий",
                   @"Венера", @"Земля", @"Марс",
                   @"Юпитер", @"Сатурн", @"Уран",
                   @"Нептун",
                   @"Луна",
                   @"Фобос", @"Деймос", @"Ио",
                   @"Европа", @"Ганимед", @"Каллисто", nil];
       NSArray *images= [[NSArray alloc] initWithObjects:@"Planet01.png",
                   @"Planet02.png", @"Planet03.png", @"Planet04.png",
                   @"Planet05.png", @"Planet06.png", @"Planet07.png",
                   @"Planet08.png",
```

```
                    @"PlanetNA.png",
                    @"PlanetNA.png", @"-", @"-",
                    @"-", @"-", @"-", nil];

        [self ShowPlanetsList: @"Все планеты": names: images];

        [names release];
        [images release];
}

- (BOOL)webView:(UIWebView*)webView
                 shouldStartLoadWithRequest:(NSURLRequest*)request
                 navigationType:(UIWebViewNavigationType)navigationType
{
    if (navigationType == UIWebViewNavigationTypeLinkClicked)
    {
        NSURL *URL = [request URL];
        NSString *link = [URL fragment];
        if ([link isEqualToString:@"ShowAll"])
        {
             [self onShowAll];
             return NO;
        }
    }
    return YES;
}

- (void)didReceiveMemoryWarning
{
    // Releases the view if it doesn't have a superview.
    [super didReceiveMemoryWarning];
}

- (void)viewDidUnload
{
    // Release any retained subviews of the main view.
}

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

@end
```

Рис. 4.15. Список планет с добавлением ссылки

Скомпилировав и запустив программу, получим список, как на рис. 4.15.

Для чередования цветов достаточно просто добавить `bgcolor="#999999"` к каждой нечетной ячейке. Перепишем код функции `ShowPlanetsList` как показано в листинге 4.21.

Листинг 4.21. Улучшенная функция вывода таблицы

```
- (void) ShowPlanetsList: (NSString*)sHeader: (NSArray*)pPlanets:
                                              (NSArray*)pImages
{
    NSString *html = [NSString stringWithFormat: @"<body>"
                      @"<h1>%@</h1>"
                      @"<table border=\"0\" cellspacing=\"0\">", sHeader];

    for(int p=0; p<[pPlanets count]; p++)
    {
        NSString *h1 = [NSString stringWithFormat:@"<td width=\"60\">"
                      @"<div align=\"center\">"
                      @"<img src=\"%@\"/></div></td>",
                             [pImages objectAtIndex:p]],
                  *h2 = [NSString stringWithFormat:@"<td width=\"240\">"
                      @"<h2>%@</h2></td>",
                             [pPlanets objectAtIndex:p]];

        if (p & 1)
            html = [NSString stringWithFormat:
```

```
                        @"%@ <tr bgcolor=\"#999999\">"
                        @"%@ %@</tr>", html, h1, h2];
                else
            html = [NSString stringWithFormat:
                   @"%@ <tr>"
                   @"%@ %@</tr>", html, h1, h2];
    }

    html = [NSString stringWithFormat:@"%@"
@"</table>"
                    @"<A HREF=\"#ShowAll\">"
                    @"Показать все"
                    @"</A>"
                    @"</body></html>", html];

    NSString *imagePath = [[NSBundle mainBundle] resourcePath];
    imagePath = [imagePath stringByReplacingOccurrencesOfString:@"/"
                                                    withString:@"//"];
    imagePath = [imagePath stringByReplacingOccurrencesOfString:@" "
                                                    withString:@"%20"];

    [mHTMLReport loadHTMLString:html baseURL:[NSURL URLWithString:
                [NSString stringWithFormat:@"file:/%@//",imagePath]]];
}
```

Чтобы белые края ячеек не мешали, были выставлены параметры таблицы `cellspacing="0"` и `border="0"`.

Окончательный результат показан на рис. 4.16.

Рис. 4.16. Вывод списка планет в HTML с чередованием цвета ячеек

Для программной обработки открытия окна с более подробным описанием планеты необходимо формировать ссылки вида `#Planet01`, `#Planet02`, а затем с помощью парсинга строки в функции `shouldStartLoadWithRequest` определить, какая именно ссылка была выбрана.

Класс `UIWebViewDelegate` позволяет разнообразить интерфейс программы с помощью функций:

- `webViewDidStartLoad` — вызывается при начале загрузки документа;
- `webViewDidFinishLoad` — вызывается в конце загрузки документа;
- `didFailLoadWithError` — вызывается в случае ошибки загрузки.

С помощью этих функций можно, например, выводить индикатор прогресса при загрузке объемного документа и убирать его, когда документ загрузится.

На этом мы закончим рассмотрение основных функций ввода-вывода и перейдем к работе с графикой с помощью сенсорного экрана.

Глава 5

Ввод и вывод данных произвольной формы

В предыдущих главах мы рассматривали различные элементы управления, их можно использовать для ввода и вывода текста, таблиц, изображений и пр. Но часто приходится иметь дело с нестандартным вводом и выводом: графики и диаграммы, рисунки и прочие подобные данные могут потребовать написания специальных функций. Рассмотрим более подробно операции ввода и вывода.

Ввод данных с помощью сенсорного экрана

Использование сенсорного экрана требует обработки трех системных сообщений: нажатие, движение и отпускание. В iOS для этого необходимо реализовать три функции:

- `touchesBegan`, отвечающую за нажатие пальца на экран;
- функцию `touchesMoved`, вызываемую многократно в процессе движения пальца;
- функцию `touchesEnded`, вызываемую при отпускании руки.

Для примера используем касание сенсорного экрана при вводе положения объекта. В качестве "объекта" нарисуем ящик, наподобие тех, что используются в известной игре Sokoban, и позволим пользователю нажатием пальца перемещать ящик по экрану.

Для создания такой программы выполним несколько шагов.

1. Создадим новый проект с названием TestUITouch.
2. Добавим изображение на окно программы с помощью готового примитива `UIImageView`.
3. Добавим функции обработки касаний и сделаем привязку кода к координатам изображения.

Первым делом создадим проект и добавим в ресурсы изображение, как показано на рис. 5.1. Перед установкой связей с помощью Interface Builder добавим в класс переменную `mImage` типа `UIImageView`. Настроим свойства изображения, как показано на рисунке (изображение ящика нужно предварительно нарисовать в любом графическом редакторе и вставить в проект).

Поле `Label` мы будем использовать для вывода координат объекта, для этого добавим в класс компонент типа `UILabel` и установим связи аналогичным образом.

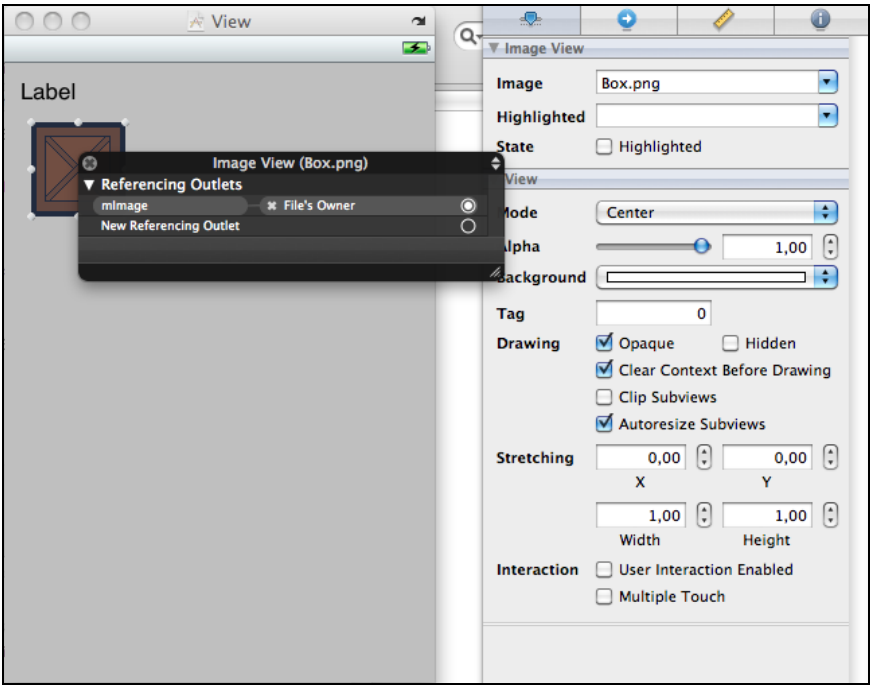


Рис. 5.1. Добавление изображения и текста к ресурсам программы

Следующим шагом необходимо сделать обработку касаний экрана. Подробное описание необходимых функций можно найти в разделе "UIResponder Class Reference" документации на сайте **http://developer.apple.com**.

Функция `touchesBegan` в качестве основного параметра принимает множество объектов типа `UITouch`, описанное как `(NSSet*)touches`. Как известно, множество может содержать более одного объекта, и это неспроста. Операционная система позволяет получать и обрабатывать множественные касания экрана, подробнее об этом будет сказано в следующей главе. Но пока вернемся к касаниям одиночным. Минимально необходимая функция, требуемая для получения одиночного касания, показана в листинге 5.1.

Листинг 5.1. Получение точки касания на экране

```
- (void)touchesBegan:(NSSet *)touches withEvent:(UIEvent *)event
{
        UITouch *touch = [touches anyObject];
        CGPoint pt = [touch locationInView:self.view];
}
```

Здесь первым делом мы получаем объект `UITouch` с помощью функции `anyObject` множества `NSSet`. Затем с помощью функции `locationInView` мы получаем координаты точки касания относительно текущего окна (для чего в функцию

передается параметр `self.view`), которые можем сохранять или анализировать. Аналогичным образом работают функции `touchesMoved` и `touchesEnded`.

Таким образом, для движения ящика по экрану нам необходимо сделать три основных действия:

1. Проверить координаты точки касания в функции `touchesBegan` и, если они находятся внутри ящика, включить режим перетаскивания.
2. В функции `touchesMoved` изменять значения координат ящика, если режим перетаскивания активен.
3. Отключить режим перетаскивания в функции `touchesEnded`.

Вставим эти функции в программу, результат показан в листинге 5.2.

Листинг 5.2. Код программы TestUITouch

Файл TestUITouchViewController.h:

```
//  TestUITouchViewController.h
//  TestUITouch
//

#import <UIKit/UIKit.h>

@interface TestUITouchViewController : UIViewController
{
       IBOutlet UIImageView     *mImage;
       IBOutlet UILabel      *mLabel;
       BOOL   bPressed;
}

@property(retain, nonatomic) UIImageView    *mImage;
@property(retain, nonatomic) UILabel            *mLabel;

@end
```

Файл TestUITouchViewController.m

```
//  TestUITouchViewController.m
//  TestUITouch
//

#import "TestUITouchViewController.h"

@implementation TestUITouchViewController
@synthesize mImage;
@synthesize mLabel;

// Implement viewDidLoad to do additional setup after loading the view,
// typically from a nib.
- (void)viewDidLoad
```

```
{
    [super viewDidLoad];

    bPressed = FALSE;
    mLabel.text = @"";
}

- (void)touchesBegan:(NSSet *)touches withEvent:(UIEvent *)event
{
    UITouch *touch = [touches anyObject];
    CGPoint pt = [touch locationInView:self.view];

    CGRect rect = [mImage frame];
    if (CGRectContainsPoint(rect, pt))
        bPressed = TRUE;
}

- (void)touchesMoved:(NSSet *)touches withEvent:(UIEvent *)event
{
    UITouch *touch = [touches anyObject];
    CGPoint pt = [touch locationInView:self.view];

    if (bPressed)
    {
        mImage.center = pt;
        mLabel.text = [NSString stringWithFormat:@"X: %d, Y: %d",
                                    (int)pt.x, (int)pt.y];
    }
}

- (void)touchesEnded:(NSSet *)touches withEvent:(UIEvent *)event
{
    bPressed = FALSE;
}

- (void)didReceiveMemoryWarning
{
    // Releases the view if it doesn't have a superview.
    [super didReceiveMemoryWarning];
}

- (void)viewDidUnload
{
}

- (void)dealloc
```

```
{
      [mImage release];
      [mLabel release];
      [super dealloc];
}

@end
```

Можно отметить использование функции `[mImage frame]`, с помощью которой мы получаем текущее положение ящика, и функции `CGRectContainsPoint`, с помощью которой проверяется, попадает ли точка в прямоугольник. Если режим перетаскивания активен, координаты изображения изменяются с помощью указания значения `mImage.center`, эти же значения используются для вывода на экран.

Если все было сделано правильно, результаты работы программы будут примерно такими, как на рис. 5.2.

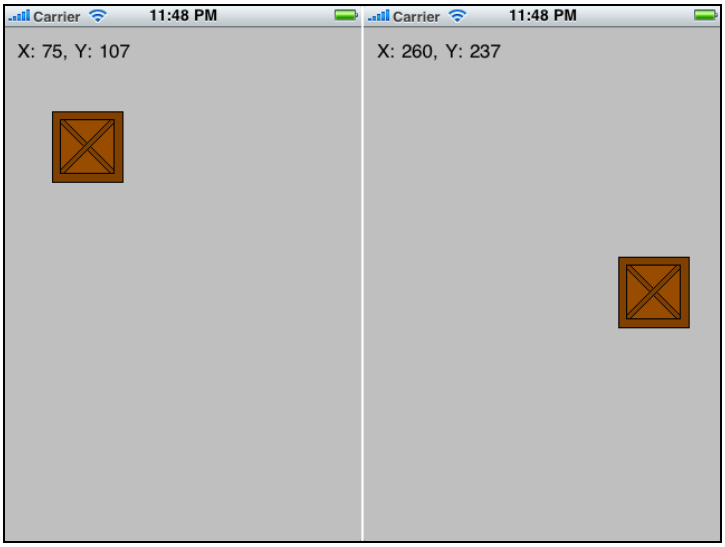


Рис. 5.2. Результаты работы программы TestUITouch

Как можно видеть, обработка касаний экрана довольно проста, следующим шагом рассмотрим вывод данных произвольной формы.

Вывод данных с помощью графических примитивов

Рисование поверх окна выполнить несколько сложнее, чем ввод данных, и так как рисовать непосредственно поверх главного окна невозможно, необходимо создать отдельный класс, наследуемый от `UIView`.

Рассмотрим для примера вывод графиков и гистограмм в программе. Для того чтобы добавить функции вывода графики в свою программу, необходимо сделать следующее:

1. Добавить в проект класс, наследуемый от `UIView`.
2. Добавить компонент этого класса в окно программы.
3. Переопределить функцию `drawRect` в созданном классе для обеспечения необходимого вывода.

Рассмотрим эти шаги подробнее. Первым делом с помощью XCode создадим новый класс, наследуемый от `UIView`, как показано на рис. 5.3.

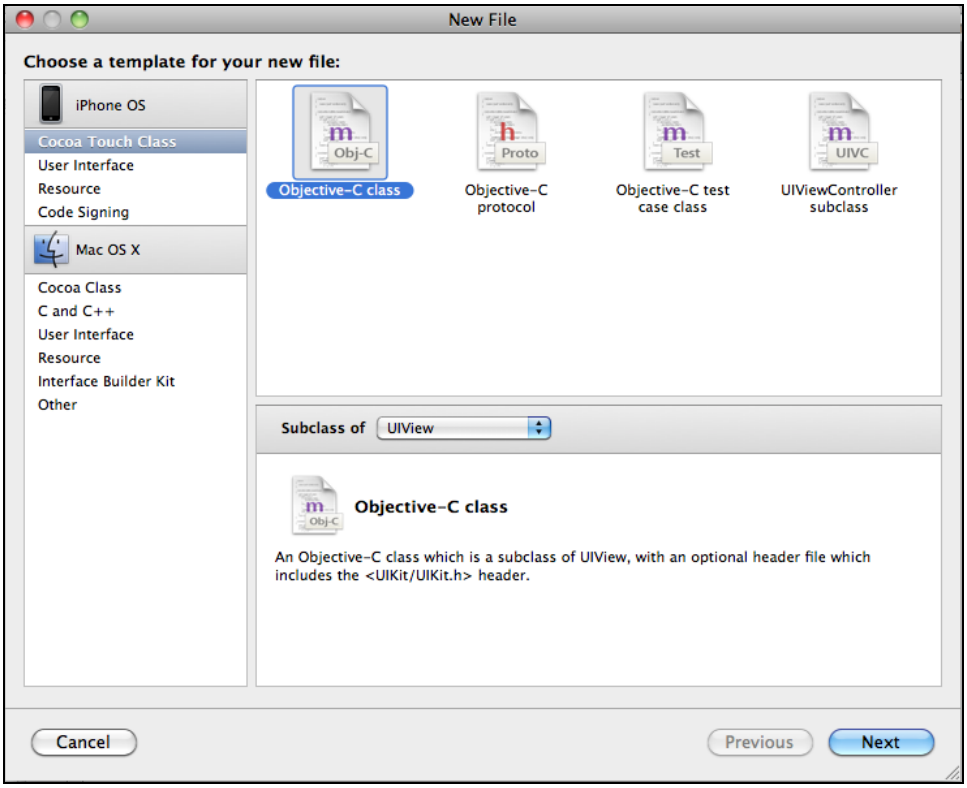


Рис. 5.3. Создание наследуемого класса

Создадим класс `TestUIDrawOutput`. Добавим в окно компонент `UIView` и две кнопки для переключения режимов. Также добавим в класс главного окна новый объект типа `TestUIDrawOutput`. Однако прежде необходимо указать класс объекта, как показано на рис. 5.4, в противном случае Interface Builder не даст установить связь между объектами различных классов. Для указания класса выберем компонент `UIView`, откроем самую правую вкладку свойств и в поле класса введем `TestUIDrawOutput`. Теперь уже можно установить связи стандартным образом, и, если все сделано правильно, свойства компонента должны соответствовать рис. 5.4.

Также добавим в основной класс программы обработчики кнопок, отвечающие за переключение режимов между графиком и гистограммой, их код мы напишем позже.

После того как мы создали класс `TestUIDrawOutput`, XCode автоматически сгенерировал два файла, в нашем примере это TestUIDraw.h и TestUIDraw.m. Рассмотрим эти файлы подробнее. Основной и наиболее интересной для нас будет функция `drawRect`, которая пока что не содержит ничего полезного.

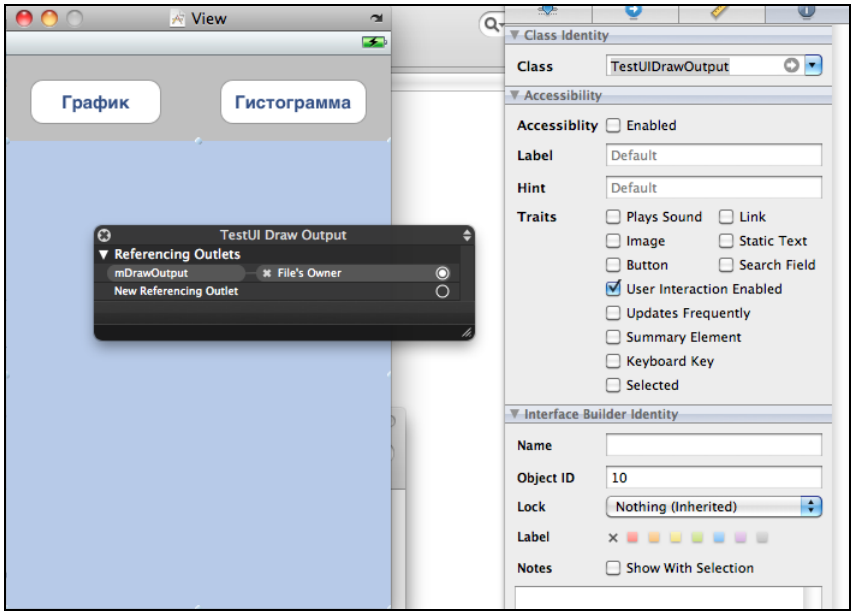


Рис. 5.4. Связывание класса `TestUIDrawOutput` в программе

Первым шагом нарисуем оси, необходимые для графиков, для чего добавим в функцию код, показанный в листинге 5.3.

Листинг 5.3. Рисование осей в функции `drawRect`

```
- (void)drawRect:(CGRect)rect
{
    CGContextRef ctx = UIGraphicsGetCurrentContext();
    CGRect frame_rect = self.frame;

    int margin = 12,  right = frame_rect.size.width - margin,
                      bottom = frame_rect.size.height - margin;
    // Линии
    CGContextSetRGBStrokeColor(ctx, 0, 0, 0, 1.0);
    CGContextSetRGBFillColor(ctx, 0, 0, 0, 1.0);

    CGContextBeginPath(ctx);
```

```
    CGContextMoveToPoint(ctx, margin, bottom);
    CGContextAddLineToPoint(ctx, margin, margin);
    CGContextMoveToPoint(ctx, margin,bottom);
    CGContextAddLineToPoint(ctx, right, bottom);
    CGContextStrokePath(ctx);

    // Текст
    {
        NSString *s_x = @"X", *s_y = @"Y", *s_z = @"0";

        UIFont *font = [UIFont fontWithName:@"Helvetica" size:10];

        CGRect out_rect1 = CGRectMake(right, bottom, 100, 100);
        [s_x drawInRect:out_rect1 withFont:font
                lineBreakMode:UILineBreakModeClip
                alignment:UITextAlignmentLeft];

        CGRect out_rect2 = CGRectMake(15, 5, 100, 100);
        [s_y drawInRect:out_rect2 withFont:font
                lineBreakMode:UILineBreakModeClip
                alignment:UITextAlignmentLeft];

        CGRect out_rect3 = CGRectMake(5, bottom, 100, 100);
        [s_z drawInRect:out_rect3 withFont:font
             lineBreakMode:UILineBreakModeClip
             alignment:UITextAlignmentLeft];
    }
}
```

В этом коде можно выделить несколько важных моментов:

- текущий графический контекст определяется с помощью функции `UIGraphicsGetCurrentContext`, в дальнейшем он используется для рисования (здесь можно привести аналогию с используемыми в Windows классами `CClientDC` и `CPaintDC`);
- текущие размеры объекта определяются с помощью свойства `self.frame`, рисование осуществляется в относительных координатах, чтобы не изменять код рисования при изменении размеров компонента, например при повороте экрана или при необходимости поставить его в другое место окна.

Совокупность функций `CGContextBeginPath`, `CGContextMoveToPoint` и `CGContextAddLineToPoint` позволяет создать одиночную или ломаную линию. Далее ее можно либо отобразить с помощью функции `CGContextStrokePath`, либо закрасить с помощью функции `CGContextFillPath`. Для вывода текста используется функция `drawInRect` класса `NSString`. У этой функции есть два важных параметра: `font`, с помощью которого мы можем задавать шрифт и размеры текста, и `alignment`, с помощью которого можно указать выравнивание.

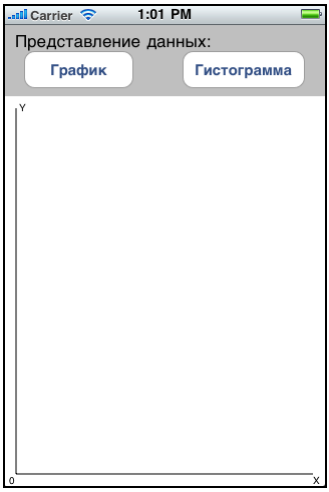


Рис. 5.5. Результат вывода компонента `TestUIDrawOutput`

Теперь можно скомпилировать программу, и если все было сделано правильно, результат будет похож на рис. 5.5.

Теперь можно сделать полноценный вывод графической информации. Добавим в класс `TestUIDrawOutput` переменную для хранения текущего режима вывода и две функции `setModeBar` и `setModeGraph`, которые будут переключать режим (листинг 5.4).

Листинг 5.4. Окончательная версия класса `TestUIDrawOutput`

Файл TestUIDraw.h:

```
//
//  TestUIDraw.h
//  TestUITouch
//

#import <UIKit/UIKit.h>

@interface TestUIDrawOutput : UIView
{
        CGPoint mPt1, mPt2;
        BOOL    bPressed;

        #define MODE_BAR    0
        #define MODE_GRAPH  1
        int     nCurrentMode;
}

- (void) doInit;
```

```
- (void) setModeBar;
- (void) setModeGraph;

@end
```

Файл TestUIDraw.m

```
//
//  TestUIDraw.m
//  TestUITouch
//
#import "TestUIDraw.h"

@implementation TestUIDrawOutput

- (id)initWithFrame:(CGRect)frame
{
    if ((self = [super initWithFrame:frame]))
    {
        // Initialization code
    }
    return self;
}

- (void) doInit
{
    bPressed = FALSE;
    nCurrentMode = MODE_GRAPH;
}

- (void) setModeBar
{
     nCurrentMode = MODE_BAR;
     [self setNeedsDisplay];
}

- (void) setModeGraph
{
    nCurrentMode = MODE_GRAPH;
    [self setNeedsDisplay];
}

// Only override drawRect: if you perform custom drawing.
// An empty implementation adversely affects performance
// during animation.
- (void)drawRect:(CGRect)rect
{
```

```
CGContextRef ctx = UIGraphicsGetCurrentContext();
CGRect frame_rect = self.frame;

// 1) Оси
int margin = 12,  right = frame_rect.size.width - margin,
                  bottom = frame_rect.size.height - margin;
// Линии
CGContextSetRGBStrokeColor(ctx, 0, 0, 0, 1.0);
CGContextSetRGBFillColor(ctx, 0, 0, 0, 1.0);

CGContextBeginPath(ctx);
CGContextMoveToPoint(ctx, margin, bottom);
CGContextAddLineToPoint(ctx, margin, margin);
CGContextMoveToPoint(ctx, margin,bottom);
CGContextAddLineToPoint(ctx, right, bottom);
CGContextStrokePath(ctx);
// Текст
{
    NSString *s_x = @"X", *s_y = @"Y", *s_z = @"0";

    UIFont *font = [UIFont fontWithName:@"Helvetica" size:10];

    CGRect out_rect1 = CGRectMake(right, bottom, 100, 100);
    [s_x drawInRect:out_rect1 withFont:font
                lineBreakMode:UILineBreakModeClip
                alignment:UITextAlignmentLeft];

    CGRect out_rect2 = CGRectMake(15, 5, 100, 100);
    [s_y drawInRect:out_rect2 withFont:font
                lineBreakMode:UILineBreakModeClip
                alignment:UITextAlignmentLeft];

    CGRect out_rect3 = CGRectMake(5, bottom, 100, 100);
    [s_z drawInRect:out_rect3 withFont:font
                lineBreakMode:UILineBreakModeClip
                alignment:UITextAlignmentLeft];
}


// 2) Данные
int data_array[] = { 2, 4, 10, 12, 16, 4, 3, 7, 11, 12, 13, 8, 5 };
int count = sizeof(data_array)/sizeof(data_array[0]);

#define XtoPos(x) (margin + 20*(x+1))
#define YtoPos(y) (bottom - 1 - 20*y)
```

```
// График
if (nCurrentMode == MODE_GRAPH)
{
    CGContextSetRGBFillColor(ctx, 0.0, 0.0, 0.0, 1.0);

    CGContextSetRGBStrokeColor(ctx, 1, 0, 0, 1.0);

    // Линии
    CGContextBeginPath(ctx);
    CGContextMoveToPoint(ctx, XtoPos(0), YtoPos(data_array[0]));

    for(int p=0; p<count; p++)
    {
        int x = XtoPos(p), y = YtoPos(data_array[p]);
        CGContextAddLineToPoint(ctx, x, y);
    }
    CGContextStrokePath(ctx);
    // Круги
    const int raduis = 10;
    for(int p=0; p<count; p++)
    {
        int x = XtoPos(p), y = YtoPos(data_array[p]);

        CGContextFillEllipseInRect(ctx,
              CGRectMake(x-raduis/2, y-raduis/2, raduis, raduis));
    }
}
// Гистограмма
if (nCurrentMode == MODE_BAR)
{
    for(int p=0; p<count; p++)
    {
        CGContextSetRGBFillColor(ctx, 0.5, 1.0, 0.5, 1.0);
        CGContextSetRGBStrokeColor(ctx, 0, 0, 0, 1.0);

        // Fill
        int x1 = XtoPos(p), x2 = XtoPos(p+1),
            y1 = YtoPos(0), y2 = YtoPos(data_array[p]);
        CGContextBeginPath(ctx);
        CGContextMoveToPoint(ctx, x1, y1);
        CGContextAddLineToPoint(ctx, x1, y2);
        CGContextAddLineToPoint(ctx, x2, y2);
        CGContextAddLineToPoint(ctx, x2, y1);
        CGContextFillPath(ctx);
```

```
            // Stroke
            CGContextBeginPath(ctx);
            CGContextMoveToPoint(ctx, x1, y1);
            CGContextAddLineToPoint(ctx, x1, y2);
            CGContextAddLineToPoint(ctx, x2, y2);
            CGContextAddLineToPoint(ctx, x2, y1);
            CGContextStrokePath(ctx);

            // Значения
            CGContextSetRGBFillColor(ctx, 0.0, 0.0, 0.0, 1.0);

            NSString *data = [NSString stringWithFormat:@"%d",
                                        data_array[p]];
            UIFont *font = [UIFont fontWithName:@"Helvetica" size:10];
            CGRect out_rect1 = CGRectMake(x1, y2, x2 - x1, 100);
            [data drawInRect:out_rect1 withFont:font
                                lineBreakMode:UILineBreakModeClip
                                alignment:UITextAlignmentCenter];
        }
    }
}

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

@end
```

Код довольно прост, хотя имеет большую длину. Стоит отметить вызов функции `[self setNeedsDisplay]` принудительной перерисовки экрана после нажатия кнопки.

Заключительным шагом необходимо добавить обработчики кнопок в класс `TestUIDrawViewController`, как показано в листинге 5.5.

Листинг 5.5. Функции переключения режимов

```
- (IBAction) onModeGraph:(id)sender
{
        [mDrawOutput setModeGraph];
}

- (IBAction) onModeBar:(id)sender
{
        [mDrawOutput setModeBar];
}
```

Естественно, переменная `mDrawOutput` должна быть корректно описана как свойство (`property`) и связана в Interface Builder, полный текст класса `TestUIDrawViewController` не показан для экономии места, его можно найти в прилагаемом к книге примере TestUIDrawSrc.zip.

Если все было сделано правильно, результат будет примерно таким, как на рис. 5.6.

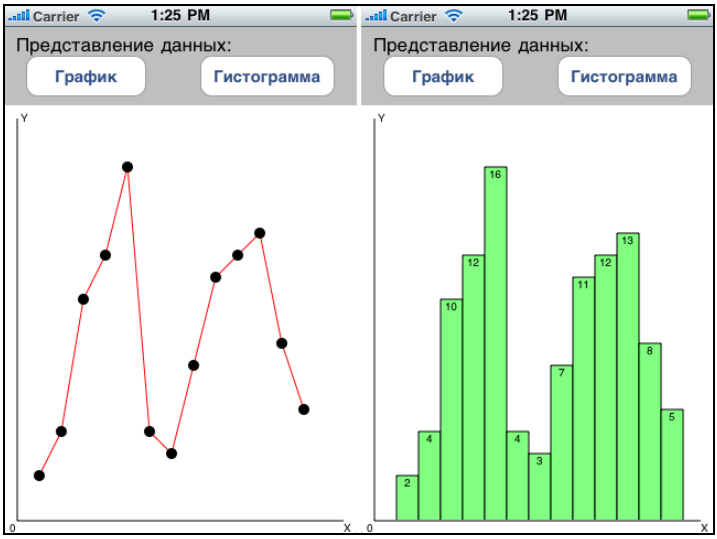


Рис. 5.6. Окончательный результат работы программы

На этом мы закончим рассмотрение основных возможностей операционной системы iOS и перейдем к более сложным функциям: работе с акселерометром, сетью Интернет и технологией multi-touch, чему будет посвящена следующая глава.

Глава 6

Интерфейс пользователя — расширенные возможности

Одно из отличий операционной системы iOS от других, имеющихся на рынке — дополнительные возможности взаимодействия устройства с пользователем, такие как автоматический поворот экрана с помощью датчика ориентации и поддержка множественных касаний экрана. Использованию этих технологий будет посвящена глава.

Ввод данных — использование технологии multi-touch

В предыдущей главе мы уже рассматривали ввод данных и функции `touchesBegan`, `touchesMoved` и `touchesEnded`. И как было замечено, входным параметром этой функции является множество, способное хранить несколько объектов. Таким образом, для того чтобы переделать предыдущий проект и добавить поддержку множественных касаний, нужно совсем немного. Для примера добавим в предыдущую программу возможность изменения размеров ящика при его касании и растягивании двумя пальцами.

Во-первых, откроем редактор ресурсов и проверим, установлены ли для главного окна флажки **User Interaction Enabled** и **Multiple Touch**, как показано на рис. 6.1.

Теперь мы можем изменить функцию `touchesBegan`. Отличие будет лишь в том, что вместо одного касания `UITouch` с помощью функции `anyObject` мы будем использовать все множество точек.

Коды новой и, для сравнения, старой функции показаны в листинге 6.1.

Листинг 6.1. Получение точек касания на экране — старый и новый варианты

```
// старый вариант
- (void)touchesBegan:(NSSet *)touches withEvent:(UIEvent *)event
{
     UITouch *touch = [touches anyObject];
     CGPoint pt = [touch locationInView:self.view];

     CGRect rect = [mImage frame];
     if (CGRectContainsPoint(rect, pt))
          bPressed = TRUE;
```

```
}

// новый вариант
- (void)touchesBegan:(NSSet *)touches withEvent:(UIEvent *)event
{
     NSSet *allTouches = [event allTouches];
     int count = [allTouches count];

     for(int i=0; i<count; i++)
     {
          UITouch *touch = [[allTouches allObjects] objectAtIndex:i];
          CGPoint pt = [touch locationInView:self.view];

          CGRect rect = [mImage frame];
          if (CGRectContainsPoint(rect, pt))
          {
               bPressed = TRUE;
               break;
          }
     }
}
```

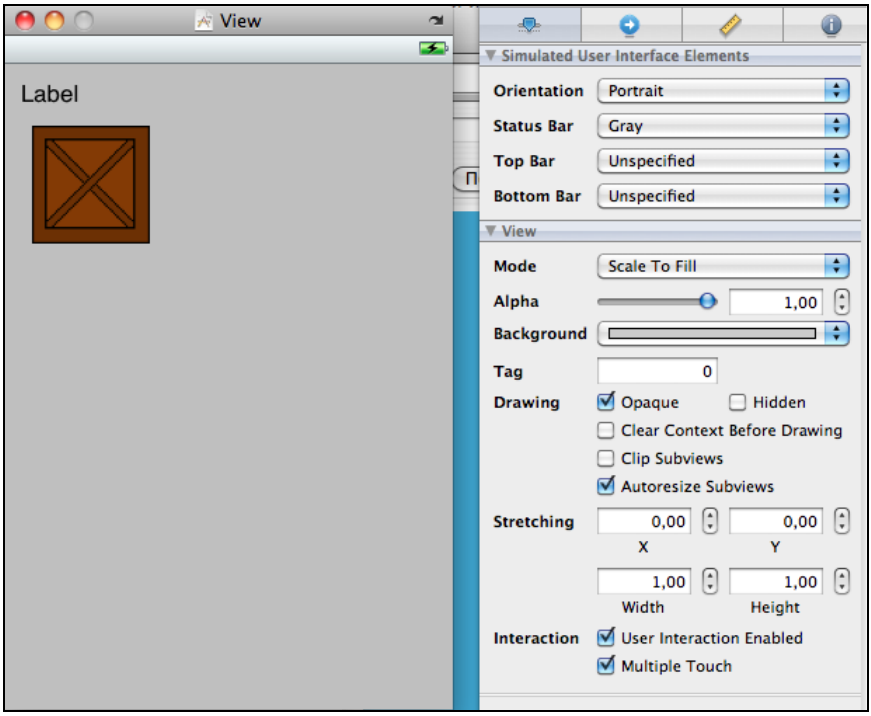


Рис. 6.1. Свойство **Multiple Touch** для окна

Принцип остался тот же: мы имеем массив элементов `UITouch` вместо одного, а функции `touchesMoved` и `touchesEnded` действуют аналогично. Зная это, можем написать окончательный текст программы, как в листинге 6.2.

Листинг 6.2. Движение объекта с использованием Multi-touch

Файл TestUITouchViewController.h:

```
//  TestUITouch
//

#import <UIKit/UIKit.h>

@interface TestUITouchViewController : UIViewController
{
        IBOutlet UIImageView    *mImage;
        IBOutlet UILabel              *mLabel;
        BOOL   bPressed;
}

@property(retain, nonatomic) UIImageView    *mImage;
@property(retain, nonatomic) UILabel              *mLabel;

@end
```

Файл TestUITouchViewController.m:

```
#import "TestUITouchViewController.h"

@implementation TestUITouchViewController
@synthesize mImage;
@synthesize mLabel;

#define min(a,b) (a<b?a:b)

// Implement viewDidLoad to do additional setup after loading the view,
// typically from a nib.
- (void)viewDidLoad
{
    [super viewDidLoad];

    bPressed = FALSE;
    mLabel.text = @"";
}

- (void)touchesBegan:(NSSet *)touches withEvent:(UIEvent *)event
{
    NSSet *allTouches = [event allTouches];
```

```
    int count = [allTouches count];

    for(int i=0; i<count; i++)
    {
        UITouch *touch = [[allTouches allObjects] objectAtIndex:i];
        CGPoint pt = [touch locationInView:self.view];

        CGRect rect = [mImage frame];
        if (CGRectContainsPoint(rect, pt))
        {
            bPressed = TRUE;
            break;
        }
    }
}

- (void)touchesMoved:(NSSet *)touches withEvent:(UIEvent *)event
{
    NSSet *allTouches = [event allTouches];
    int count = [allTouches count];

    if (!bPressed) return;

    if (count == 1)
    {
        // одиночное касание
        UITouch *touch = [[allTouches allObjects] objectAtIndex:0];
        CGPoint pt = [touch locationInView:self.view];

        mImage.center = pt;
        mLabel.text = [NSString stringWithFormat:@"X: %d, Y: %d",
                                                 (int)pt.x, (int)pt.y];
    } else
    if (count == 2)
    {
        // масштабирование
        UITouch  *touch1 = [[allTouches allObjects] objectAtIndex:0],
                 *touch2 = [[allTouches allObjects] objectAtIndex:1];
        CGPoint pt1 = [touch1 locationInView:self.view],
                pt2 = [touch2 locationInView:self.view];

        float px = min(pt1.x, pt2.x), py = min(pt1.y, pt2.y),
              cx = (pt1.x + pt2.x)/2, cy = (pt1.y + pt2.y)/2,
              size_x = fabs(pt2.x - pt1.x), size_y = fabs(pt2.y - pt1.y);


        mImage.frame = CGRectMake(px, py, size_x, size_y);
```

```
        mLabel.text = [NSString stringWithFormat:
                           @"X: %d, Y: %d, Size: %dx%d",
                           (int)cx, (int)cy, (int)size_x, (int)size_y];
    }
}

- (void)touchesEnded:(NSSet *)touches withEvent:(UIEvent *)event
{
    bPressed = FALSE;
}

- (void)didReceiveMemoryWarning
{
    // Releases the view if it doesn't have a superview.
    [super didReceiveMemoryWarning];
}

- (void)viewDidUnload
{
}

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

@end
```

Для изменения положения картинки мы используем свойство `mImage.center`, для масштабирования — определяем новый прямоугольник из координат обеих точек. Поскольку мы заранее не знаем, какая из точек будет верхней, а какая нижней, верхняя точка прямоугольника определяется как минимум из обеих координат (напомним, что координаты увеличиваются сверху вниз и слева направо), а размеры прямоугольника определяются как абсолютная величина разности двух значений. Если этого не сделать, мы можем получить прямоугольник с отрицательной шириной или высотой, который, естественно, не отобразится на экране. Если все было сделано правильно, изображение должно изменяться, как показано на рис. 6.2.

У программы есть очевидный недостаток: перетаскивается центр объекта `mImage.center = pt`, даже если пользователь начал передвигать объект за край. Для более логичного движения нужно в функции `touchesBegan` запоминать расстояние от точки нажатия до центра ящика и в функции `touchesMoved` прибавлять к центральной точке это смещение. В качестве дополнительного упражнения читатели могут исправить код самостоятельно.

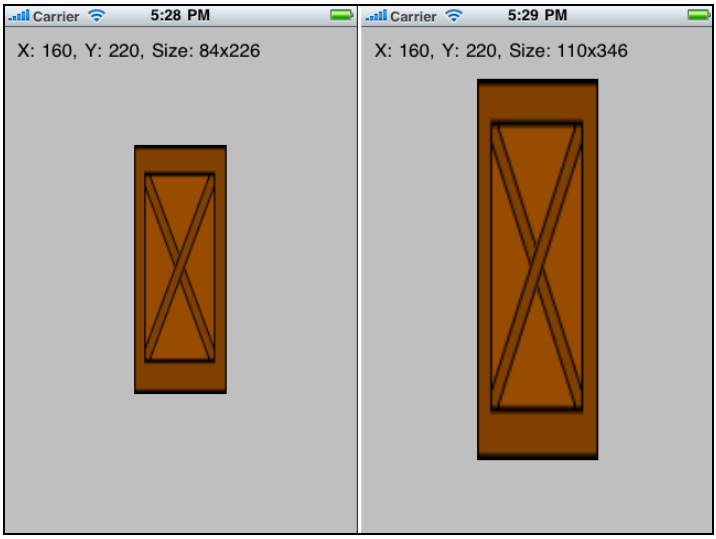


Рис. 6.2. Изменение размеров объекта

Аналогичным методом можно выполнять и другие операции преобразования, например, одиночное касание использовать для сдвига, а двойное — для масштабирования и поворота. Важно помнить, что в отличие от мыши или стилуса, нажатие пальцем имеет меньшую точность, и это также нужно учитывать. К сожалению, в некоторых приложениях, изначально не рассчитанных на сенсорные экраны, кнопки навигации часто делают так близко расположенными, что нажатие их на iPhone или iPad часто приводит к попаданию "не туда".

На этом мы закончим рассмотрение использования сенсорного экрана и перейдем к другим способам ввода данных.

Ввод данных — использование акселерометра

За обмен данными с акселерометром отвечает класс `UIAccelerometer`, подробнее можно прочитать о нем в разделе документации "UIAccelerometer Class Reference".

Для того чтобы получать данные о наклоне устройства от акселерометра в программе, нужно сделать следующее:

1. Добавить в класс окна функцию `accelerometer:didAccelerate`, описанную в разделе "UIAccelerometerDelegate Protocol Reference".
2. Активировать получение сообщений акселерометра с помощью функции `UIAccelerometer:setDelegate`.

Возвращаемые акселерометром значения *x*, *y* и *z* — это величины, пропорциональные проекции наклона устройства на соответствующие оси, и прямого отношения к координатам они не имеют. Таким образом, мы можем использовать в программе наклон устройства, но не можем отследить его параллельное перемещение.

Рассмотрим практический пример — сделаем программу, отображающую модель авиагоризонта, плоскость которого всегда параллельна плоскости земли. Необходимо будет выполнить несколько шагов.

1. Подготовить два изображения авиагоризонта, представляющих его корпус (статическое изображение) и индикатор (динамическое изображение).
2. Разместить изображения в окне так, чтобы статическое оказалось поверх динамического, и привязать динамическое изображение к переменной типа `UIImageView`.
3. Добавить функцию `accelerometer:didAccelerate`, в которой нужно изменять угол изображения в соответствии с наклоном устройства.

Создадим в Adobe Photoshop два изображения, как показано на рис. 6.3.

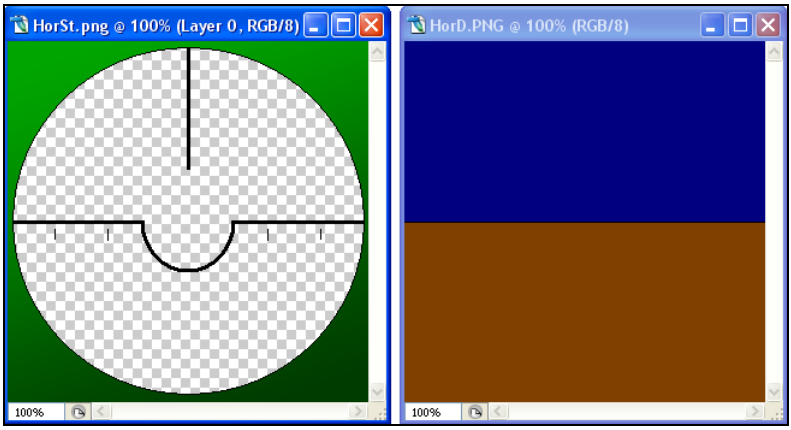


Рис. 6.3. Изображения "индикатора" и "корпуса"

Обратим внимание на прозрачные области модели слева: они нужны, чтобы правый рисунок индикатора был виден сквозь "корпус". Сохраним эти изображения в формате PNG (этот формат поддерживает прозрачность).

Теперь добавим в класс переменную типа `UIImageView` и разместим динамическое (правое) изображение. Осуществим привязку добавленного изображения к переменной с помощью Interface Builder. Зададим свойства второго изображения, как показано на рис. 6.4.

Поле **Background** (Фон) установлено в значение **Clear color** и снят флажок **Opaque** (Непрозрачный): одна картинка должна быть видна сквозь другую, как на рисунке. Совместим изображения, чтобы одно находилось точно поверх другого. Верхний элемент `UILabel` (с текстом `0.0`) мы будем использовать для отладки, чтобы значения было видно во время наклона устройства.

После того как подготовительные операции завершены, можно написать код, непосредственно использующий акселерометр. В описании протокола `UIAccelerometerDelegate` есть лишь одна функция `accelerometer:didAccelerate`, которая имеет вид, показанный в листинге 6.3.

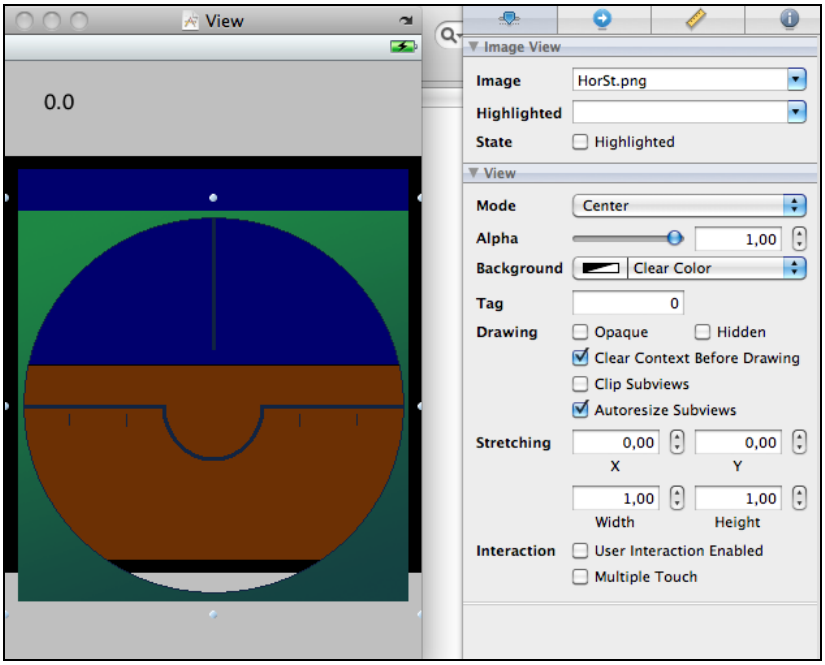


Рис. 6.4. Свойства прозрачности для изображения

Листинг 6.3. Функция `didAccelerate`

```
- (void)accelerometer:(UIAccelerometer *)accelerometer
                        didAccelerate:(UIAcceleration *)acceleration
{
    float x = acceleration.x, y = acceleration.y, z = acceleration.z;
    // вывод на экран
    mLabel.text = [NSString stringWithFormat:@"x:%f, y:%f", x, y];
}
```

Как можно видеть из описания "UIAcceleration Class Reference", этот класс имеет три проекции *x*, *y* и *z* на соответствующие оси.

Глядя на экран и наклоняя устройство в разные стороны, получаем следующие значения:

- *x*=0, *y*=1 — устройство расположено вертикально;
- *x*=0, *y*=1 — устройство повернуто влево;
- *x*=1, *y*=0 — устройство повернуто вправо;
- *x*=1, *y*=0 — устройство повернуто вниз.

Если нарисовать оси и подписать значения, то мы получим координатный круг, где нужный угол можно определить как арксинус соответствующей координаты. Зная это, можем написать окончательный код нашей программы в листинге 6.4.

Листинг 6.4. Класс `TestAccelViewController`

Файл TestAccelViewController.h:

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

@interface TestAccelViewController : UIViewController<UIAccelerometerDelegate>
{
       IBOutlet UILabel      *mLabel;
       IBOutlet UIImageView  *mHorDynamic;

       float fAcceleromterX, fAcceleromterY;
}

- (void) UpdateImage;

@property(retain, nonatomic) UILabel              *mLabel;
@property(retain, nonatomic) UIImageView   *mHorDynamic;

@end
```

Файл TestAccelViewController.m:

```
//  TestAccelViewController.m

#import "TestAccelViewController.h"

@implementation TestAccelViewController
@synthesize mLabel;
@synthesize mHorDynamic;

// Implement viewDidLoad to do additional setup after loading the view,
// typically from a nib.
- (void)viewDidLoad
{
    [super viewDidLoad];

     // "Включить" акселерометр
     [[UIAccelerometer sharedAccelerometer] setUpdateInterval: 1.0/10.0];
     [[UIAccelerometer sharedAccelerometer] setDelegate:self];
}

- (void)accelerometer:(UIAccelerometer *)accelerometer
                          didAccelerate:(UIAcceleration *)acceleration
{
    fAcceleromterX = acceleration.x;
    fAcceleromterY = acceleration.y;
```

```
    [self UpdateImage];
}

- (void) UpdateImage
{
    mLabel.text = [NSString stringWithFormat:@"x:%f, y:%f",
                        fAcceleromterX, fAcceleromterY];

    float angle = 0;
    if (fAcceleromterX != 0.0f)
    {
        if (fAcceleromterX < 0)
            angle = M_PI/2 - atan(fAcceleromterY/fAcceleromterX);
                else
            angle = 3*M_PI/2 - atan(fAcceleromterY/fAcceleromterX);
    }

    mHorDynamic.transform = CGAffineTransformMakeRotation(angle);
}

- (void)didReceiveMemoryWarning
{
    // Releases the view if it doesn't have a superview.
    [super didReceiveMemoryWarning];
}

- (void)viewDidUnload
{
     // "Выключить" акселерометр
     [[UIAccelerometer sharedAccelerometer] setDelegate:nil];
}


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

@end
```

Здесь стоит остановиться на нескольких моментах.

- Функция `UIAccelerometer:setDelegate` вызывается при старте программы и при ее завершении, для этого используются функции `viewDidLoad` и `viewDidUnload` соответственно. Функция `setUpdateInterval` используется для

указания временного интервала, с которым в программу будут приходить сообщения от акселерометра.

- Получаемый (в радианах) угол используется для указания свойства `transform` объекта `UIImageView`, с помощью которого можно задать любое графическое преобразование. Для указания трансформации поворота используется функция `CGAffineTransformMakeRotation`.
- Приведенный пример будет работать в iOS 3.0, но может не работать в 4.0 при свертывании программы в фоновый режим и возврате из него. Для корректной работы с возвратом из фонового режима нужно вставить соответствующие обработчики в функции `applicationWillResignActive` и `applicationDidEnterBackground` класса `AppDelegate`. Желающие могут добавить требуемые строки кода самостоятельно.
- Акселерометр не работает в симуляторе, функция `didAccelerate` вообще не вызывается. Странно, что сделано именно так, ведь симулятор поддерживает повороты на 90°, и сообщения вполне могли бы приходить. Но что есть — то есть, отладить приведенную программу удастся лишь на реальном устройстве.

Если все было сделано правильно, результат работы программы при наклоне устройства будет похож на рис. 6.5. Некоторым недостатком программы является то, что при повороте устройства края нижнего изображения становятся видны из-под верхнего. Исправить это можно, либо сделав прозрачные края у нижнего изображения, либо увеличив размеры верхнего так, чтобы оно покрывало большую область. Изменение прозрачности нижнего изображения с помощью Adobe Photoshop показано на рис. 6.6.

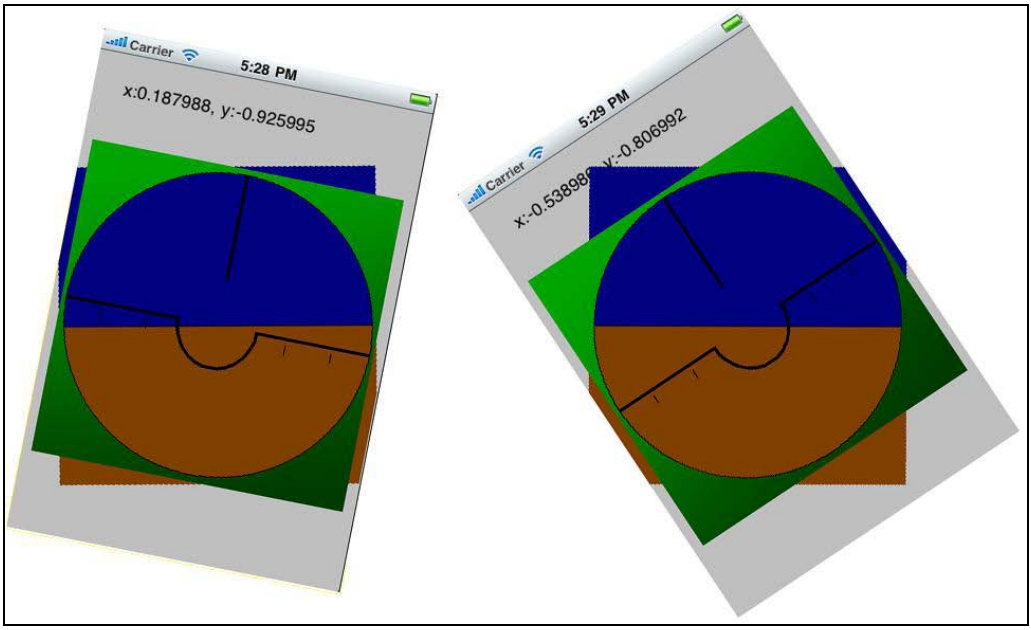


Рис. 6.5. Работа авиагоризонта при наклоне устройства

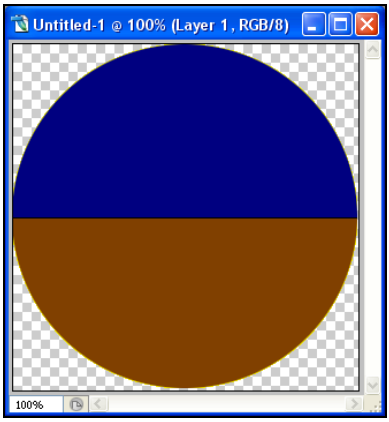


Рис. 6.6. Указание прозрачных границ в Adobe Photoshop

Если обрезать края нижнего изображения таким образом, мешать они уже не будут. Для этого следует создать новое изображение в Adobe Photoshop, указав прозрачность слоя (background — transparent), скопировать имеющееся изображение, выделить круг нужного размера, затем инвертировать выделение и удалить ненужное. Получившийся результат следует сохранить в формате PNG вместо старого файла.

Теперь при запуске программы никаких торчащих углов больше не будет.

На этом мы закончим изучение акселерометра, желающие провести более подробные эксперименты могут скачать из App Store бесплатную программу AccelGraph, реализующую графическое отображение данных акселерометра.

Загрузка данных из сети Интернет

Постоянный доступ в сеть Интернет стал обыденностью, и все больше приложений активно используют сетевое соединение для получения и отображения данных (курс валют, прогноз погоды, предупреждения о пробках на дороге). Поэтому немаловажно уметь получать данные из сети и выводить их в программе.

Перейдем сразу к практическому примеру, будем загружать текущее состояние погоды по ссылке **http://www.google.com/ig/api?weather=CITY**, где **CITY** это требуемый город. Например, если открыть в браузере ссылку **http://www.google.com/ig/api?weather=Moscow**, то мы получим текст, показанный в листинге 6.5.

Листинг 6.5. Данные прогноза погоды от Google

```
<?xml version="1.0"?>
<xml_api_reply version="1">
<weather module_id="0" tab_id="0" mobile_row="0" mobile_zipped="1" row="0"
section="0" >
<forecast_information>
  <city data="Moscow"/>
```

```
  <postal_code data="Moscow"/>
  <latitude_e6 data=""/>
  <longitude_e6 data=""/>
  <forecast_date data="2011-01-20"/>
  <current_date_time data="2011-01-20 18:00:00 +0000"/>
  <unit_system data="SI"/>
</forecast_information>
<current_conditions>
   <condition data="Небольшой снег"/>
   <temp_f data="14"/>
   <temp_c data="-10"/>
   <humidity data="Влажность: 83 %"/>
   <icon data="/ig/images/weather/flurries.gif"/>
   <wind_condition data="Ветер: ЮВ, 1 м/с"/>
</current_conditions>
<forecast_conditions>
   <day_of_week data="Чт"/>
   <low data="-16"/>
   <high data="-9"/>
   <icon data="/ig/images/weather/snow.gif"/>
   <condition data="Ливневый снег"/>
</forecast_conditions>
<forecast_conditions>
…
</forecast_conditions>
</weather>
</xml_api_reply>
```

Запомним структуру этого текста, она нам пригодится в дальнейшем для обработки получаемых и выделения нужных данных. Для примера будем выводить текущую температуру воздуха в выбранном городе.

Нам необходимо разобраться с тем, как загружать данные. В iOS существует два способа получения данных.

- Синхронная загрузка — простой способ. Мы делаем запрос, функция сразу (точнее говоря, после небольшой паузы) возвращает полученный ответ. Этот способ имеет большой недостаток, связанный с тем, что система "зависает" до тех пор, пока функция загрузки не закончит работу.
- Асинхронная загрузка — этот способ более сложен в реализации, зато он и более гибкий. Мы запускаем функцию загрузки и можем выполнять какие-либо другие действия, например показать индикатор "песочные часы". В тот момент, когда данные будут получены, программа получит от системы сообщение.

Очевидно, что первый способ удобен в тех случаях, когда надо получить небольшой блок данных по заведомо быстрому каналу, и наоборот, второй способ удобнее в загрузке больших объемов данных или для медленных соединений.

Рассмотрим реализацию обеих методов. Для начала создадим приложение с полем ввода `UITextField` и свяжем его с переменной `m_cCity` названия города.

Первый способ загрузки показан в листинге 6.6.

Листинг 6.6. Синхронная загрузка данных

```
NSString *city = m_cCity.text;
NSString *url_path = [NSString
      stringWithFormat:@"http://www.google.com/ig/api?weather=%@", city];

NSURL *url = [NSURL URLWithString: url_path];
NSString *data = [[[NSString alloc]
       initWithContentsOfURL:url
       encoding:NSASCIIStringEncoding
       error:nil] autorelease];
if (data != nil)
{
       // Обработка полученной строки
       …
}
```

Мы формируем строку запроса в переменной `url_path`, которая используется для инициализации объекта `NSURL`. Используя функцию `initWithContentsOfURL`, мы загружаем непосредственно в строку требуемые данные: поскольку операционная система iOS является интернет-ориентированной, даже класс работы со строками имеет встроенную функцию загрузки данных из сети. В Windows-классах типа `CString` ничего такого автору не попадалось.

Вторым способом является асинхронная загрузка, которая выполняется несколько сложнее. Для того чтобы загрузить данные, необходимо:

1. Создать объекты `NSURLRequest` и `NSURLConnection`.
2. Добавить функцию `connection:didReceiveResponse`, которая будет вызываться в начале загрузки данных.
3. Добавить функцию `connection:didReceiveData`, которая будет вызываться во время загрузки.
4. Добавить функции `connectionDidFinishLoading` и `didFailWithError`, которые будут вызываться при окончании загрузке или при возникновении ошибки (например, из-за несуществующего адреса).

Функция `didReceiveData` может вызываться несколько раз, и данные принимаются фрагментами. Поэтому разработчику необходимо хранить получаемые данные в буфере памяти.

Объединим все в работоспособную программу, код которой показан в листинге 6.7.

Листинг 6.7. Код программы синхронной и асинхронной загрузки данных

Файл TestInternetViewController.h:

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

@interface TestInternetViewController : UIViewController
{
       IBOutlet UITextField *m_cCity;
       IBOutlet UILabel       *m_cLabel;

       NSMutableData        *m_cWeatherData;
}

- (IBAction) onGetData1:(id)sender;
- (IBAction) onGetData2:(id)sender;
- (void) SetWeatherFromString: (NSString*)str;

@property(retain, nonatomic) UILabel        *m_cLabel;
@property(retain, nonatomic) UITextField *m_cCity;

@end
```

Файл TestInternetViewController.m:

```
#import "TestInternetViewController.h"

@implementation TestInternetViewController
@synthesize m_cLabel;
@synthesize m_cCity;

// Implement viewDidLoad to do additional setup after loading the view,
// typically from a nib.
- (void)viewDidLoad
{
    [super viewDidLoad];

    m_cWeatherData = [[NSMutableData data] retain];
    m_cCity.text = @"Moscow";
}

- (IBAction) onGetData1:(id)sender
{
    NSString *city = m_cCity.text;
    NSString *url_path = [NSString
      stringWithFormat:@"http://www.google.com/ig/api?weather=%@", city];

    NSURL *url = [NSURL URLWithString: url_path];
```

```
    NSString *data = [[[NSString alloc] initWithContentsOfURL:url
                                        encoding:NSASCIIStringEncoding
                                        error:nil] autorelease];
    if (data != nil)
    {
         [self SetWeatherFromString:data];
    } else
    {
        m_cLabel.text = @"error";
    }
}

- (IBAction) onGetData2:(id)sender
{
    NSString *city = m_cCity.text;
    NSString *url_path = [NSString
      stringWithFormat:@"http://www.google.com/ig/api?weather=%@", city];

    NSURL *url = [NSURL URLWithString: url_path];

    NSURLRequest *request = [NSURLRequest requestWithURL:url];
    [[[NSURLConnection alloc] initWithRequest:request delegate:self]
         autorelease];
}

- (void)connectionDidFinishLoading:(NSURLConnection *)connection
{
    if ([m_cWeatherData length] > 0)
    {
        NSString *data = [[[NSString alloc]
                        initWithBytes:[m_cWeatherData bytes]
                        length:[m_cWeatherData length]
                        encoding: NSASCIIStringEncoding] autorelease];
        if (data != nil)
        {
             [self SetWeatherFromString:data];
        }
    }
}

- (void)connection:(NSURLConnection *)connection
                        didReceiveResponse:(NSURLResponse *)response
{
     [m_cWeatherData setLength:0];
}
```

```
- (void)connection:(NSURLConnection *)connection
                        didReceiveData:(NSData *)data
{
     [m_cWeatherData appendData:data];
}

- (void)connection:(NSURLConnection *)connection
                        didFailWithError:(NSError *)error
{
    m_cLabel.text = @"Error";
}

- (void) SetWeatherFromString: (NSString*)str
{
}

- (void)didReceiveMemoryWarning
{
    // Releases the view if it doesn't have a superview.
    [super didReceiveMemoryWarning];
}

- (void)viewDidUnload
{
}


- (void)dealloc
{
     [m_cWeatherData release];
     [m_cLabel release];
     [m_cCity  release];

     [super dealloc];
}

@end
```

Программа состоит из обработчиков двух нажатий кнопок, активирующих оба вида загрузки, и поля `UILabel`, в которое мы затем будем выводить результат. Для хранения данных во время загрузки используется массив `NSMutableData`, добавляются они в функции `didReceiveData`, полностью загруженная строка сохраняется как `NSString` в функции `connectionDidFinishLoading`.

На этом программу можно было бы закончить, но ведь мы еще не доделали функцию загрузки данных. Обратимся к листингу 6.5. Как можно видеть из струк-

туры XML, данные температуры находятся внутри блока `temp_c`, который, в свою очередь, находится внутри блока `current_conditions`. Напишем функцию, которая выделяет подстроку из строки по заданным маркерам начала и конца. Код этой функции показан в листинге 6.8.

Листинг 6.8. Функция загрузки подстроки

```
- (NSString*) GetSubString: (NSString*)sStr:
                            (NSString*)sDivider1:
                            (NSString*)sDivider2
{
     NSString *s_ret = nil;

     NSRange tr1 = [sStr rangeOfString:sDivider1];
     if (tr1.location != NSNotFound)
     {
          NSRange sr = NSMakeRange(tr1.location + tr1.length,
                              [sStr length] - tr1.location - tr1.length);

          NSRange tr2 = [sStr rangeOfString:sDivider2
                              options:NSCaseInsensitiveSearch range:sr];
          if (tr2.location != NSNotFound)
          {
               NSRange dr = NSMakeRange(tr1.location + tr1.length,
                              tr2.location - tr1.location - tr1.length);
               s_ret = [sStr substringWithRange:dr];
          }
     }

     return s_ret;
}
```

Эта функция нужна для того, чтобы выделять нужные строки из XML. Например, для строки `str = "Данные<DATA>10</DATA>"` вызов функции с параметрами `[self GetSubString: str: @"<DATA>": @"</DATA>"]` должен вернуть строку `10`.

Парсинг данных текущей температуры становится довольно простым (листинг 6.9).

Листинг 6.9. Парсинг данных для получения температуры

```
- (void) SetWeatherFromString: (NSString*)str
{
      NSString *data = [self GetSubString:str :@"<current_conditions>":
                                              @"</current_conditions>"];
      if (data != nil)
      {
```

```
            NSString *tempr = [self GetSubString:data :@"<temp_c" :@"/"];
            if (tempr != nil)
            {
                  tempr = [self GetSubString:tempr :@"data=\"" :@"\""];

                  m_cLabel.text = tempr;
            }
      }
}
```

Мы сначала запрашиваем блок `current_conditions`, затем `temp_c`, затем выделяем окончательную подстроку (со строки `data=`). Теперь мы можем вставить эту функцию в наш код, и программа работоспособна (рис. 6.7).

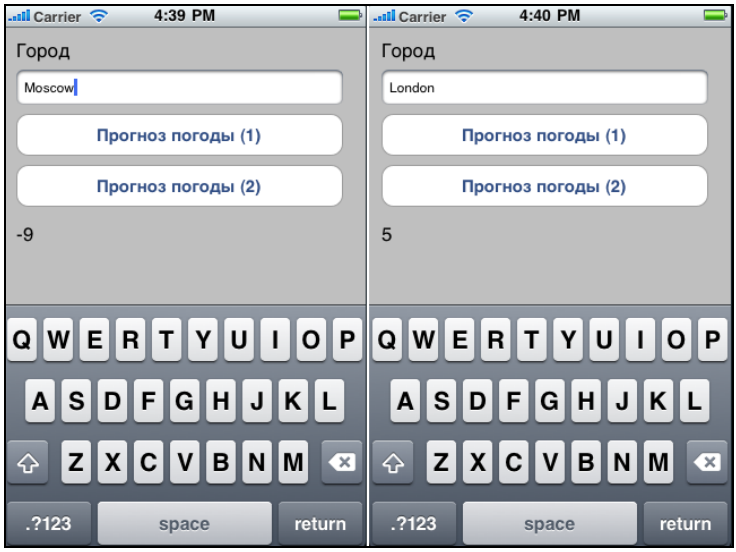


Рис. 6.7. Отображение данных из сети Интернет в программе

Желающие могут самостоятельно доработать эту программу, сделав вывод графических значков погоды, ссылки на которые хранятся в поле `icon`, и прогноза погоды на несколько дней.

Поддержка вертикальной и горизонтальной ориентации экрана

Ориентация экрана может быть обусловлена как спецификой данных (некоторые таблицы удобнее смотреть на широком экране, а фотографии — наоборот), так и личными предпочтениями пользователя. Поддержка разворота экрана настоятельно рекомендуется Apple для коммерческих приложений. Даже если специфика

приложения подразумевает только горизонтальную или вертикальную ориентацию, все равно есть минимум два варианта положения устройства — кнопкой вверх или вниз (или слева/справа).

Для того чтобы обеспечить поддержку разворота экрана в программе, нужно сделать следующее.

1. Добавить в проект функцию `shouldAutorotateToInterfaceOrientation`, которая должна возвращать `TRUE` для тех видов ориентации, которые поддерживаются в программе.
2. Настроить для компонентов привязку к требуемым сторонам экрана.

Функция `shouldAutorotateToInterfaceOrientation` уже описывалась *в главе 2*, более того, она априори есть в каждом проекте в виде комментария, поэтому сразу перейдем к описанию второго шага.

Для указания привязки компонентов следует выбрать требуемый компонент и в настройках компонента открыть вкладку **Size&Position**, как показано на рис. 6.8. В разделе **Autosizing** можно указать, как будет меняться положение компонента относительно экрана.

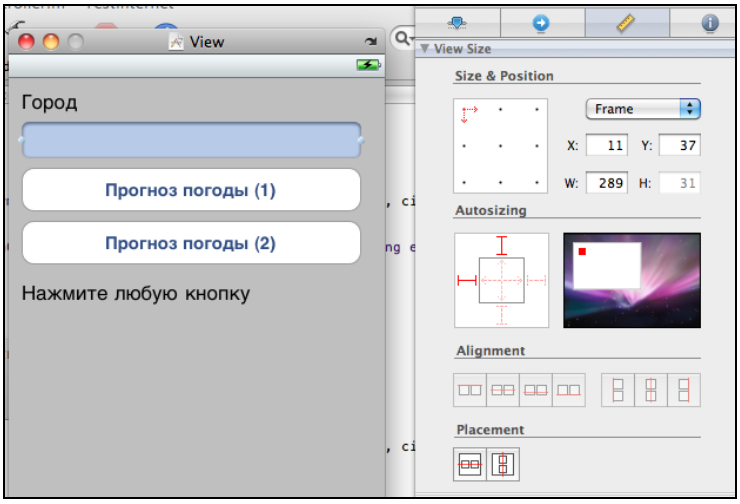


Рис. 6.8. Привязка размеров компонента к размеру экрана

Доступны несколько вариантов:

- компонент выравнивается слева;
- компонент выравнивается справа;
- компонент сдвигается относительно требуемого края;
- компонент растягивается относительного требуемого края.

Все настраивается визуально в редакторе ресурсов и довольно понятно. Однако если от компонентов требуется нестандартное поведение, можно задать координаты вручную, переопределив функцию `didRotateFromInterfaceOrientation`, как показано в листинге 6.10.

Листинг 6.10. Пример программного указания положения компонентов

```
- (void)didRotateFromInterfaceOrientation:
                        (UIInterfaceOrientation)fromInterfaceOrientation
{
      UIDeviceOrientation ori = [[UIDevice currentDevice] orientation];

       if (ori == UIDeviceOrientationLandscapeLeft ||
           ori == UIDeviceOrientationLandscapeRight)
      {
            // Hor
            m_cLabel.frame  = CGRectMake(20, 20, 163, 21);
       } else
       if (ori == UIDeviceOrientationPortrait ||
            ori == UIDeviceOrientationPortraitUpsideDown)
       {
            // Ver
            m_cLabel.frame  = CGRectMake(20, 40, 133, 21);
       }
}
```

Рассмотрим практический пример. В программе, отображающей погоду, добавим возможность работы во всех положениях экрана. Выполним два шага.

1. Раскомментируем функцию `shouldAutorotateToInterfaceOrientation` и установим возвращаемое значение в `TRUE` для всех положений экрана.

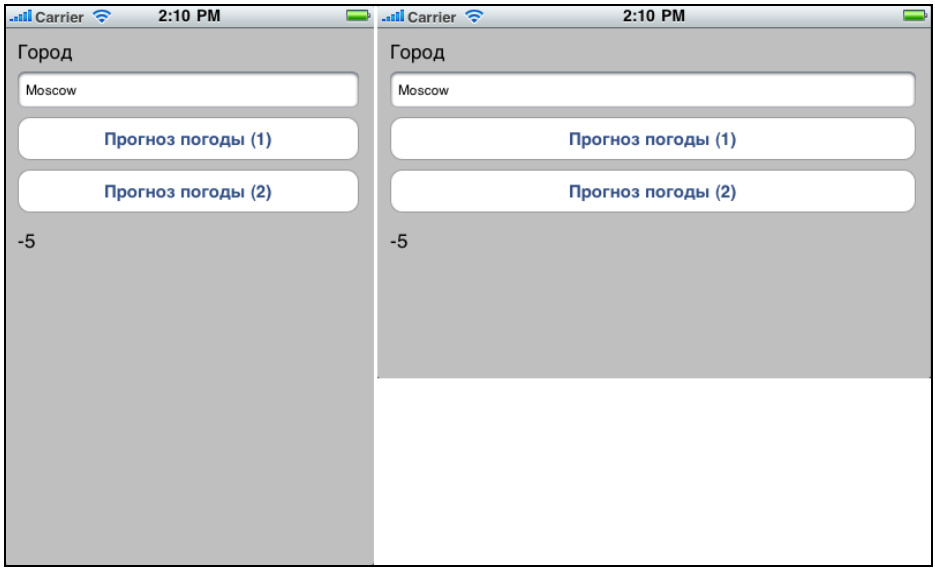


Рис. 6.9. Программа в горизонтальной и вертикальной ориентации экрана

2. Откроем редактор ресурсов и установим следующие свойства: для меток с названием города и температурой выставим выравнивание слева, для поля ввода и кнопок выставим растяжение на весь экран.

   Во время настроек удобно пользоваться кнопкой поворота окна (рис. 6.9).

Стоит заметить, что поворот экрана с помощью функции `shouldAutorotateToInterfaceOrientation` возможен лишь для приложений, использующих `UIViewController`. При использовании приложений другого типа поворот экрана придется реализовать самостоятельно (например, в OpenGL указанием требуемой матрицы поворота).

Создание полноэкранных приложений

Согласно рекомендациям Apple, стоит трижды подумать, прежде чем делать приложение полноэкранным: в верхней части окна отображается полезная для владельца устройства информация (часы, уровень сигнала сотовой связи, заряд батареи). Однако для некоторых приложений (особенно игровых) это может быть целесообразно. Для реализации полноэкранного режима возможны два способа:

- установить переменную `UIStatusBarHidden` в файле info.plist, в таком случае приложение будет автоматически развертываться на весь экран при запуске;
- использовать функцию `UIApplication:setStatusBarHidden`, которая позволяет переключать режим во время работы программы.

Для примера добавим возможность переключения режима экрана в программу, отображающую авиагоризонт, показанную на рис. 6.5. Добавим кнопку на экран, с помощью Interface Builder привяжем ее к правому верхнему краю экрана. В обработчике нажатия кнопки вставим следующий код (листинг 6.11).

Листинг 6.11. Переключение режимов экрана из программы

```
- (IBAction) onSwitchFullscreen:(id)sender
{
        static BOOL fs = FALSE;
        fs = !fs;

        [[UIApplication sharedApplication] setStatusBarHidden:fs
                                          withAnimation:NO];
}
```

Результат работы программы показан на рис. 6.10.

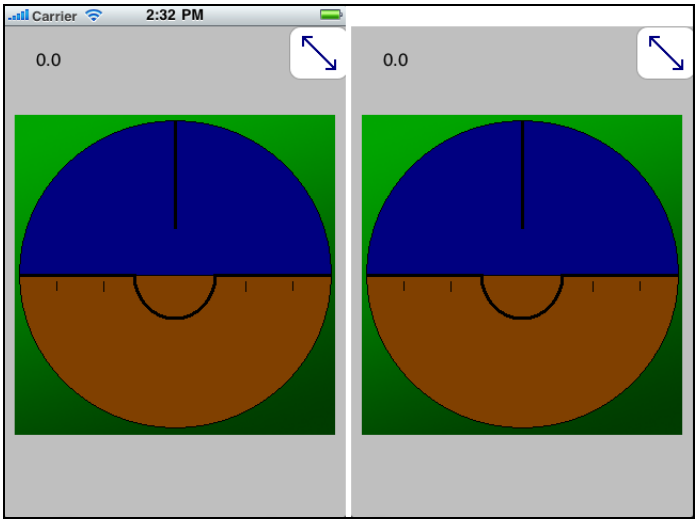


Рис. 6.10. Переключение полноэкранного режима в программе

Верхняя панель статуса действительно исчезает, но это не совсем то, что нам нужно. Необходимо скорректировать размеры окна в соответствии с новым размером экранной области (20 пикселов панели статуса). Исправленная функция показана в листинге 6.12.

Листинг 6.12. Окончательный вариант функции переключения режимов

```
- (IBAction) onSwitchFullscreen:(id)sender
{
      static BOOL fs = FALSE;
      fs = !fs;

      CGRect main_rect = [[UIScreen mainScreen] bounds];
      const int top_height = 20;

      [[UIApplication sharedApplication] setStatusBarHidden:fs
                                        withAnimation:NO];
      if (fs)
      {
            [self.view setFrame:
                  CGRectMake(0, 0, main_rect.size.width,
                                   main_rect.size.height)];
      } else
      {
            [self.view setFrame:
```

```
                    CGRectMake(0, top_height,
                               main_rect.size.width,
                               main_rect.size.height - top_height)];
        }
}
```

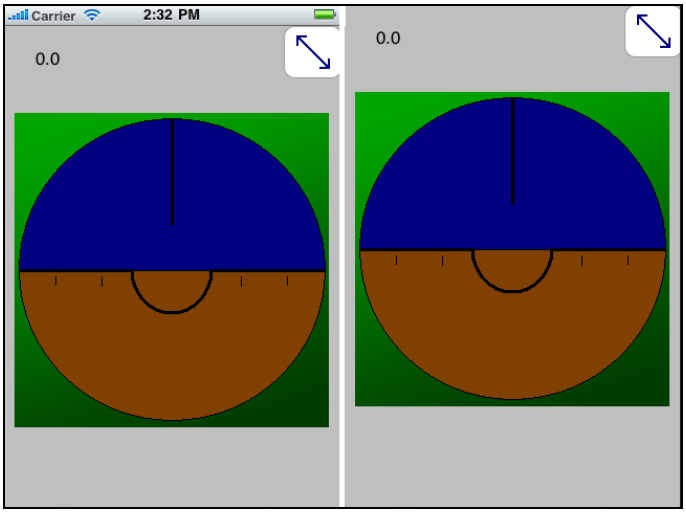


Рис. 6.11. Переключение полноэкранного режима в программе

После такой доработки окно переключается корректно, как показано на рис. 6.11.

На этом мы закончим рассмотрение полноэкранного режима и перейдем к проигрыванию звуковых файлов.

Воспроизведение звуковых файлов

В iOS SDK предусмотрено сразу несколько способов воспроизведения звука. В зависимости от сложности поставленной задачи пользователь может выбрать один из трех основных методов.

- Для воспроизведения коротких (не более 30 с) звуков — нажатия кнопки, выстрела и прочих коротких фрагментов — можно использовать функцию `AudioServicesPlaySystemSound`.
- Для воспроизведения более сложных звуков можно использовать класс `AVAudioPlayer`. Он имеет бóльшие возможности по сравнению с первым способом: позволяет воспроизводить и приостанавливать звук, изменять громкость и т. п.
- При необходимости еще более сложной обработки звука можно использовать функции `AudioQueue`, позволяющие напрямую посылать данные в буфер звуковой карты. Это сложный метод программной обработки звука, и мы здесь рассматривать его не будем, ограничимся первыми двумя.

Создадим приложение для воспроизведения звуков музыкальных нот и короткого сигнала.

Использование функции *AudioServicesPlaySystemSound*

Для воспроизведения коротких звуков с помощью `AudioServicesPlaySystemSound` нужно сделать три шага:

1. Добавить необходимые звуки в ресурсы приложения в любом распространенном формате, например в виде WAV-файлов.
2. Добавить AudioToolbox framework к ресурсам программы.
3. Добавить функцию воспроизведения звука.

Для примера добавим в ресурсы проекта файл Short.wav. Функция, воспроизводящая звук, показана в листинге 6.13.

Листинг 6.13. Воспроизведение короткого звука

```
- (IBAction) onPlayShort:(id)sender
{
      // Получить путь к файлу
      NSString *path = [NSString stringWithFormat:@"%@%@",
                                [[NSBundle mainBundle] resourcePath],
                                          @"/Short.wav"];
      NSURL *file = [NSURL fileURLWithPath:path isDirectory:NO];

      // Создать объект SystemSound
      SystemSoundID soundID;
      AudioServicesCreateSystemSoundID((CFURLRef)file, &soundID);

      AudioServicesPlaySystemSound(soundID);
}
```

Мы создаем звук с помощью функции `AudioServicesCreateSystemSoundID`, затем проигрываем его с помощью вызова `AudioServicesPlaySystemSound`. Если звук планируется проигрывать большое число раз, целесообразно сделать переменную `soundID` членом класса, а `AudioServicesCreateSystemSoundID` выполнять один раз в функции `viewDidLoad`.

Использование класса *AVAudioPlayer*

Следующим способом является использование `AVAudioPlayer`. Для проигрывания звука этим методом необходимо выполнить следующие действия:

1. Добавить необходимые звуки в ресурсы приложения.
2. Подключить к приложению AVFoundation framework во избежание ошибок компоновки.
3. Для каждого звука создать объект `AVAudioPlayer` с помощью функции `AVAudioPlayer::initWithContentsOfURL`.
4. Вызвать функции `play` и `stop` для запуска или остановки воспроизведения.
5. Удалить объект `AVAudioPlayer` по окончании работы программы.

Для примера создадим два звуковых файла с нотами "До" и "Ре", добавив в проект кнопки воспроизведения. В отличие от предыдущих проектов, для каждой кнопки добавим по два обработчика: один для нажатия, другой для отпускания кнопки. С их помощью мы будем проигрывать звук, пока кнопка нажата (листинг 6.14).

Листинг 6.14. Воспроизведение звуков с помощью `AVAudioPlayer`

Файл TestSoundsViewController.h:

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

@interface TestSoundsViewController :
                          UIViewController<AVAudioPlayerDelegate>
{
       AVAudioPlayer *soundDo, *soundRe;
}

- (IBAction) onPlayDo:(id)sender;
- (IBAction) onStopDo:(id)sender;

- (IBAction) onPlayRe:(id)sender;
- (IBAction) onStopRe:(id)sender;

@end
```

Файл TestSoundsViewController.m:

```
#import <AudioToolbox/AudioServices.h>
#import "TestSoundsViewController.h"

@implementation TestSoundsViewController

// Implement viewDidLoad to do additional setup after loading the view,
// typically from a nib.
- (void)viewDidLoad
{
    [super viewDidLoad];

    NSError *error;
    NSString *exe_path = [[NSBundle mainBundle] resourcePath];
    // Звук До:
    NSURL *url1 = [NSURL fileURLWithPath:
                    [NSString stringWithFormat:@"%@/SDo.wav", exe_path]];
    soundDo = [[AVAudioPlayer alloc] initWithContentsOfURL:url1
                     error:&error];
    if (soundDo != nil)
    {
```

```
        soundDo.numberOfLoops = -1; // бесконечный цикл
        soundDo.delegate = self;
    }
    // Звук Re:
    NSURL *url2 = [NSURL fileURLWithPath:
                [NSString stringWithFormat:@"%@/SRe.wav", exe_path]];
    soundRe = [[AVAudioPlayer alloc] initWithContentsOfURL:url2
                                      error:&error];
    if (soundRe != nil)
    {
        soundRe.numberOfLoops = -1; // бесконечный цикл
        soundRe.delegate = self;
    }
}

// Override to allow orientations other than the default
// portrait orientation.
- (BOOL)shouldAutorotateToInterfaceOrientation:
                              (UIInterfaceOrientation)interfaceOrientation
{
   return (interfaceOrientation == UIInterfaceOrientationLandscapeLeft ||
           interfaceOrientation == UIInterfaceOrientationLandscapeRight);
}

- (IBAction) onPlayDo:(id)sender
{
    if (soundDo != nil)
         [soundDo play];
}

- (IBAction) onStopDo:(id)sender
{
    if (soundDo != nil)
         [soundDo stop];
}

- (IBAction) onPlayRe:(id)sender
{
    if (soundRe != nil)
         [soundRe play];
}

- (IBAction) onStopRe:(id)sender
{
    if (soundRe != nil)
```

```
        [soundRe stop];
}

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

- (void)viewDidUnload
{
}

- (void)dealloc
{
     [soundDo release];
     [soundRe release];

     [super dealloc];
}

@end
```

Опционально можно использовать протокол `AVAudioPlayerDelegate` для получения уведомления о различных событиях, например об окончании воспроизведения файла. Описание протокола можно найти в разделе "AVAudioPlayerDelegate Protocol Reference" на сайте **http://developer.apple.com**.

На этом мы закончим раздел, посвященный пользовательскому интерфейсу, и приступим к рассмотрению способов взаимодействия программы с операционной системой.

Глава 7

Взаимодействие с операционной системой

В предыдущих главах мы изучили стандартные компоненты графического интерфейса. Перейдем теперь к более близкому взаимодействию с операционной системой.

Загрузка изображений из встроенного фотоальбома

Как известно, все устройства Apple имеют встроенное приложение для просмотра фотографий, а смартфоны и планшетные компьютеры iPad 2 также имеют встроенную камеру. Поэтому периодически встает задача загрузки снимков из фотоальбома для какой-либо обработки: отправки или печати.

Как уже говорилось ранее, операционная система iOS не предоставляет прямого доступа к файлам или папкам, поэтому непосредственно обратиться к файлам в фотоальбоме мы не можем. Однако в iOS есть специальный класс `UIImagePickerController` для получения изображений из фотоальбома или встроенной камеры. Для его использования нужно сделать следующее:

1. Добавить объект класса `UIImagePickerController` и функцию-обработчик `didFinishPickingMediaWithInfo`.
2. Настроить параметры доступа к данным, указав, какие именно объекты мы хотим получать.
3. В нужный момент вызвать функцию `presentModalViewController` для получения списка изображений.
4. Получить выбранное изображение в функции обработки, добавленной на первом шаге.

Более подробное описание можно найти в разделе "UIImagePickerControllerDelegate Protocol Reference" документации на сайте Apple. Описание класса можно найти там же в разделе "UIImagePickerController Class Reference".

Рассмотрим практический пример — создадим программу, которая позволит пользователю выбрать фото из встроенного фотоальбома, выведет его на экран и сохранит в файл для дальнейшей обработки. Для того чтобы убедиться в результатах, создадим в программе отдельную кнопку для загрузки того же изображения из сохраненного файла. Перейдем к исходному коду в листинге 7.1.

Листинг 7.1. Программа загрузки и отображения данных из фотоальбома

Файл TestPhotoAlbumViewController.h:

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

@interface TestPhotoAlbumViewController :
              UIViewController<UIImagePickerControllerDelegate,
                               UINavigationControllerDelegate>
{
       UIImagePickerController *imgPicker;
       IBOutlet UIImageView *pImageView;
}

- (IBAction) onButtonBrowse:(id)sender;
- (IBAction) onButtonFile:(id)sender;
- (IBAction) onButtonClear:(id)sender;

@property (nonatomic, retain) UIImagePickerController *imgPicker;
@property (nonatomic, retain) UIImageView *pImageView;

@end
```

Файл TestPhotoAlbumViewController.m:

```
#import "TestPhotoAlbumViewController.h"

@implementation TestPhotoAlbumViewController
@synthesize imgPicker;
@synthesize pImageView;

// Implement viewDidLoad to do additional setup after loading the view,
// typically from a nib.
- (void)viewDidLoad
{
    [super viewDidLoad];

    imgPicker = [[UIImagePickerController alloc] init];
    imgPicker.delegate = self;
    imgPicker.sourceType = UIImagePickerControllerSourceTypePhotoLibrary;
}

- (IBAction) onButtonBrowse:(id)sender
{
     [self presentModalViewController:self.imgPicker animated:YES];
}

- (void)imagePickerController:(UIImagePickerController *)picker
```

```
                    didFinishPickingMediaWithInfo:(NSDictionary *)info
{
     [picker dismissModalViewControllerAnimated:YES];

     // Get image
     UIImage *img = [info
            objectForKey:@"UIImagePickerControllerOriginalImage"];
     [pImageView setImage:img];

     // Save to file
     NSString  *png_path = [NSHomeDirectory()
        stringByAppendingPathComponent:@"Documents/ImageFromAlbum.png"];

     NSData     *data = UIImagePNGRepresentation(img);
     [data writeToFile:png_path atomically:YES];
}

- (IBAction) onButtonFile:(id)sender
{
     NSString  *png_path = [NSHomeDirectory()
        stringByAppendingPathComponent:@"Documents/ImageFromAlbum.png"];

     UIImage *img = [UIImage imageWithContentsOfFile:png_path];
     if (img != nil)
           [pImageView setImage:img];
}

- (IBAction) onButtonClear:(id)sender
{
     pImageView.image = nil;
}

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

- (void)viewDidUnload
{
}

- (void)dealloc
{
     [imgPicker release];
```

```
    [pImageView release];

    [super dealloc];
}

@end
```

Как можно видеть, в функции `viewDidLoad` был создан объект класса `UIImagePickerController` и заданы значения полей `sourceType` и `delegate`. В обработчике кнопки `onButtonBrowse` использована функция активации окна. Если пользователь выбрал изображение, срабатывает функция `didFinishPickingMediaWithInfo`, одним из параметров которой является массив объектов `info`. Получив объект `UIImage` из этого массива, мы можем его загрузить в `UIImageView` с помощью функции `setImage`. Для сохранения в PNG-файл мы используем функцию `UIImagePNGRepresentation`. Для загрузки сохраненного ранее файла мы используем функцию `imageWithContentsOfFile`. Для формирования корректного пути к локальному файлу мы используем функцию `NSHomeDirectory()`, ведь в iOS записывать что-либо в папку, отличную от локальной, мы не можем.

В редакторе ресурсов Interface Builder должны быть созданы соответствующие кнопки и компонент `ImageView` (рис. 7.1).

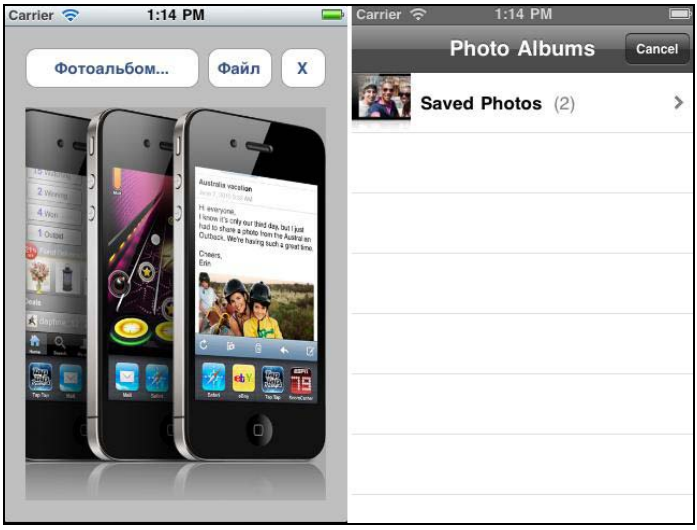


Рис. 7.1. Загрузка данных из фотоальбома

При запуске программы должно появляться окно со списком файлов фотоальбома. При выборе фотографии она должна появляться на экране. Содержимое окна можно очистить нажатием кнопки **X**, при нажатии кнопки **Файл** сохраненное изображение должно появиться снова. Если закрыть программу и открыть снова, при нажатии кнопки **Файл** изображение также должно загружаться.

Может возникнуть вопрос: как загрузить фотографии в симулятор для тестирования? Самый простой способ — это открыть встроенный в эмулятор браузер Safari и, выбрав нужную картинку, из контекстного меню (появляющегося при длительном нажатии) выбрать сохранение в локальном фотоальбоме.

Загрузка списка контактов

После рассмотрения использования фотоальбома понять принцип загрузки контактов гораздо проще. Мы не можем напрямую обратиться к телефонной книге, для этого существует класс `ABPeoplePickerNavigationController`, позволяющий пользователю выбрать нужный контакт и возвращающий нужные данные. Для обработки данных необходимо реализовать функции протокола `ABPeoplePickerNavigationControllerDelegate`, описание которого можно найти на сайте Apple. Рассмотрим практический пример — создадим программу, дающую возможность пользователю выбрать контакт и показывающую параметры этого контакта.

Создадим новое приложение, добавим в окно программы кнопку открытия списка контактов и три элемента Label, которые будут отображать полученные данные. Установим соответствующие связи с помощью Interface Builder. Подключим к программе компоненты Address Book framework и Address Book UI framework, иначе при компиляции возникнут ошибки. Рассмотрим код программы, показанный в листинге 7.2. Добавленные строки выделены жирным шрифтом.

Листинг 7.2. Программа загрузки и отображения данных из фотоальбома

Файл AddressBookViewController.h:

```
#import <AddressBook/AddressBook.h>
#import <AddressBookUI/AddressBookUI.h>

@interface AddressBookViewController :
             UIViewController<ABPeoplePickerNavigationControllerDelegate>
{
       IBOutlet UILabel *pFirstName;
       IBOutlet UILabel *pLastName;
       IBOutlet UILabel *pPhone;
}

- (IBAction) onButtonBrowse:(id)sender;

@property (nonatomic, retain) UILabel *pFirstName;
@property (nonatomic, retain) UILabel *pLastName;
@property (nonatomic, retain) UILabel *pPhone;

@end
```

Файл AddressBookViewController.m:

```
#import "AddressBookViewController.h"

@implementation AddressBookViewController
@synthesize pFirstName;
@synthesize pLastName;
@synthesize pPhone;

// Implement viewDidLoad to do additional setup after loading the view,
// typically from a nib.
- (void)viewDidLoad
{
    [super viewDidLoad];
}

- (IBAction) onButtonBrowse:(id)sender
{
    // Создать объект
    ABPeoplePickerNavigationController *picker =
          [[ABPeoplePickerNavigationController alloc] init];
    picker.peoplePickerDelegate = self;

    // Показать
    [self presentModalViewController:picker animated:YES];

    [picker release];
}

- (BOOL)peoplePickerNavigationController:
              (ABPeoplePickerNavigationController *)peoplePicker
              shouldContinueAfterSelectingPerson:(ABRecordRef)person
{

    // First name
    pFirstName.text = (NSString *)ABRecordCopyValue(person,
                                     kABPersonFirstNameProperty);

    // Last name
    pLastName.text = (NSString *)ABRecordCopyValue(person,
                                    kABPersonLastNameProperty);

    // Phone numbers
    ABMultiValueRef multi = ABRecordCopyValue(person,
                                       kABPersonPhoneProperty);
    for(CFIndex i = 0; i < ABMultiValueGetCount(multi); i++)
```

```
    {
        NSString* mobileLabel =
                  (NSString*)ABMultiValueCopyLabelAtIndex(multi, i);
        if ([mobileLabel
                  isEqualToString:(NSString*)kABPersonPhoneMobileLabel])
        {
         pPhone.text = (NSString*)ABMultiValueCopyValueAtIndex(multi, i);
        }
    }

    // remove the controller
    [peoplePicker dismissModalViewControllerAnimated:YES];

    return NO;
}

- (BOOL)peoplePickerNavigationController:
          (ABPeoplePickerNavigationController *)peoplePicker
          shouldContinueAfterSelectingPerson:(ABRecordRef)person
          property:(ABPropertyID)property
          identifier:(ABMultiValueIdentifier)identifier
{
    return NO;
}

- (void)peoplePickerNavigationControllerDidCancel:
          (ABPeoplePickerNavigationController *)peoplePicker
{
     [peoplePicker dismissModalViewControllerAnimated:YES];
}

- (void)didReceiveMemoryWarning
{
    // Releases the view if it doesn't have a superview.
    [super didReceiveMemoryWarning];
}

- (void)viewDidUnload
{
}


- (void)dealloc
{
```

```
    [pLastName release];
    [pFirstName release];
    [pPhone release];

    [super dealloc];
}

@end
```

Мы создали объект в функции `onButtonBrowse`, затем при выборе пользователем контакта вызывается функция `shouldContinueAfterSelectingPerson`. В этой функции одним из параметров является поле `person`, из которого мы получаем поля `First Name` и `Last Name` с помощью функций `ABRecordCopyValue`.

Отдельно стоит обратить внимание на получение телефонных номеров. Вначале мы получаем список всех телефонов в виде элемента `ABMultiValueRef`. Затем мы просматриваем этот список до тех пор, пока не найдем нужный нам номер, имеющий тип `kABPersonPhoneMobileLabel`. И наконец, мы закрываем окно списка контактов с помощью вызова функции `dismissModalViewControllerAnimated`. Эта же функция в обработчике `peoplePickerNavigationControllerDidCancel` вызывается в том случае, если пользователь нажал кнопку **Cancel**.

Если все было сделано правильно, интерфейс программы будет выглядеть, как на рис. 7.2.

Может возникнуть и обратная задача программного добавления нового контакта в список. Для этого существуют функции `ABAddressBookAddRecord` и `ABAddressBookSave`. Для примера рассмотрим использование этих функций в листинге 7.3.

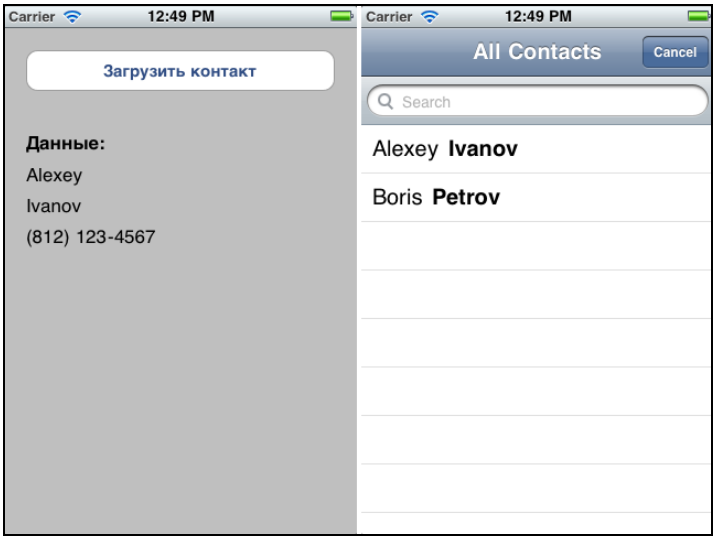


Рис. 7.2. Загрузка данных из списка контактов

Листинг 7.3. Добавление нового контакта в телефонную книгу

```
ABRecordRef aRecord = ABPersonCreate();

CFErrorRef  anError = NULL;
ABRecordSetValue(aRecord, kABPersonFirstNameProperty,
                          CFSTR("Alex"), &anError);
ABRecordSetValue(aRecord, kABPersonLastNameProperty,
                          CFSTR("Alexandrov"), &anError);
if (anError != NULL) // error
{
    return;
}

ABAddressBookRef addressBook = ABAddressBookCreate();

CFErrorRef error = NULL;
BOOL isAdded = ABAddressBookAddRecord(addressBook, aRecord, &error);

if (isAdded)
{
    // OK
    error = NULL;
    ABAddressBookSave(addressBook, &error);
}

CFRelease(aRecord);
CFRelease(addressBook);
```

Этот фрагмент кода добавляет новый контакт. На этом мы закончим рассмотрение работы с телефонной книгой.

Получение состояния батареи

Состояние батареи актуально для полноэкранных приложений, работающих долгое время: программ для чтения книг, RSS-лент новостей, музыкальных плееров и пр. Некоторые владельцы телефонов любят производить отдельный уход за батареей, делать полные циклы заряда-разряда. И хотя полезность, скорее всего, состоит разве что в психологическом комфорте владельца батареи, существуют программы, облегчающие этот процесс, например, Battery Go Plus. В таких программах анализ состояния батареи также актуален.

Для получения этих данных необходимо сделать следующее:

1. Активировать получение состояния батареи с помощью функции `UIDevice:setBatteryMonitoringEnabled`.

2. Добавить обработчики `UIDeviceBatteryLevelDidChangeNotification` и `UIDeviceBatteryStateDidChangeNotification`.
3. В функции-обработчике получить состояние батареи с помощью вызова `UIDevice:batteryLevel` и `UIDevice:batteryState`.

Необходимый код показан в листинге 7.4. Перед запуском программы необходимо добавить соответствующие ресурсы с помощью Interface Builder.

Листинг 7.4. Программа вывода состояния батареи

Файл TestBatteryViewController.h:

```
//  TestBatteryViewController.h
//

#import <UIKit/UIKit.h>

@interface TestBatteryViewController : UIViewController
{
       IBOutlet UILabel *pStatusLabel;
       IBOutlet UILabel *pLevelLabel;
}

- (void)showBatteryStatus;

@property(nonatomic, retain) UILabel *pStatusLabel;
@property(nonatomic, retain) UILabel *pLevelLabel;

@end
```

Файл TestBatteryViewController.m:

```
#import "TestBatteryViewController.h"

@implementation TestBatteryViewController
@synthesize pStatusLabel;
@synthesize pLevelLabel;

// Implement viewDidLoad to do additional setup after loading the view, //
typically from a nib.
- (void)viewDidLoad
{
     [super viewDidLoad];

     [[UIDevice currentDevice] setBatteryMonitoringEnabled:YES];
     [self showBatteryStatus];

     [[NSNotificationCenter defaultCenter] addObserver:self
            selector:@selector(showBatteryStatus)
            name:UIDeviceBatteryLevelDidChangeNotification object:nil];
```

```
     [[NSNotificationCenter defaultCenter] addObserver:self
            selector:@selector(showBatteryStatus)
            name:UIDeviceBatteryStateDidChangeNotification object:nil];
}

- (void)showBatteryStatus
{
    float level = [[UIDevice currentDevice] batteryLevel]*100;
    pLevelLabel.text = [NSString stringWithFormat:@"%.1f", level];

    int state = [[UIDevice currentDevice] batteryState];
    switch (state)
    {
    case UIDeviceBatteryStateUnknown:
        pStatusLabel.text = @"Battery: unknown";
        break;
    case UIDeviceBatteryStateUnplugged:
        pStatusLabel.text = @"Battery: unplugged";
        break;
    case UIDeviceBatteryStateCharging:
        pStatusLabel.text = @"Battery: charging";
        break;
    case UIDeviceBatteryStateFull:
        pStatusLabel.text = @"Battery: full";
        break;
    }
}

- (void)didReceiveMemoryWarning
{
    // Releases the view if it doesn't have a superview.
    [super didReceiveMemoryWarning];
}

- (void)viewDidUnload
{
     [[UIDevice currentDevice] setBatteryMonitoringEnabled:NO];
}

- (void)dealloc
{
     [pStatusLabel release];
     [pLevelLabel release];

     [super dealloc];
}

@end
```

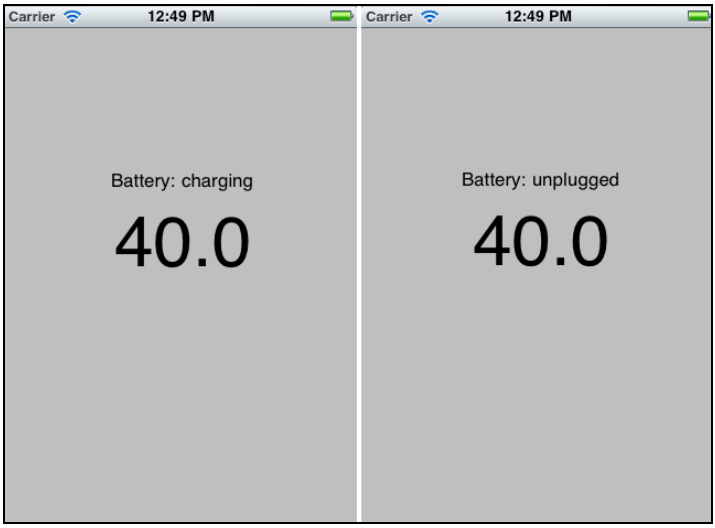


Рис. 7.3. Отображение состояния батареи

Функция получения уведомлений о состоянии батареи не работает в симуляторе, тестировать программу можно только на реальном устройстве (рис. 7.3).

Работа с файлами

Как уже говорилось, программа может сохранять и загружать данные лишь из своей локальной папки. Всего возможны два основных варианта действий:

- загрузка файлов, хранящихся в ресурсах программы;
- чтение и запись файлов в подкаталоге Documents вместе с программой.

Первый случай простой. Мы можем хранить файлы любого типа прямо в ресурсах программы и загружать их по необходимости. Это могут быть любые данные, например, изображения, звуки или XML-файлы. Во втором случае мы можем записывать и читать файлы в папку Documents, пользователь может видеть эти данные с помощью iTunes и сохранять для дальнейшей работы.

Выполним практический пример: сделаем в программе возможность записи и чтения текстовых файлов. Создадим новый проект, поместим в его ресурсы файл с именем file.txt. В редакторе ресурсов добавим текстовое поле и две кнопки для чтения и записи набранного текста в файл. Добавим третью кнопку, которую используем для вывода на экран содержимого файла file.txt. Готовый текст программы приведен в листинге 7.5.

Листинг 7.5. Чтение и запись текстовых файлов

Файл TestFilesViewController.h:

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

@interface TestFilesViewController : UIViewController
```

```
{
       IBOutlet UITextView   *m_cText;
}
@property (nonatomic, retain) UITextView   *m_cText;

- (IBAction) onButtonSave: (id)     sender;
- (IBAction) onButtonLoad: (id)     sender;
- (IBAction) onButtonLoadResource: (id)    sender;

@end
```

Файл TestFilesViewController.m:

```
#import "TestFilesViewController.h"

@implementation TestFilesViewController
@synthesize m_cText;

- (IBAction) onButtonSave: (id)     sender
{
    NSArray *arrayPaths = NSSearchPathForDirectoriesInDomains(
                            NSDocumentDirectory,
                            NSUserDomainMask, YES);
    NSString *dir = [arrayPaths objectAtIndex:0];
    NSString *path = [dir stringByAppendingString:@"/1.txt"];

    NSError *error;
    NSString *text = m_cText.text;
    BOOL res = [text writeToFile:path atomically:YES
                    encoding:NSUTF8StringEncoding error:&error];
    if (res)
        m_cText.text = @"Сохранено";
}

- (IBAction) onButtonLoad: (id)     sender
{
    NSArray *arrayPaths = NSSearchPathForDirectoriesInDomains(
                            NSDocumentDirectory,
                            NSUserDomainMask, YES);
    NSString *dir = [arrayPaths objectAtIndex:0];
    NSString *path = [dir stringByAppendingString:@"/1.txt"];

    NSError *error;
    NSString *str = [NSString stringWithContentsOfFile:path
                        encoding:NSUTF8StringEncoding error:&error];


    if (str != nil)
```

```
        m_cText.text = str;
            else
        m_cText.text = @"file not found";
}

- (IBAction) onButtonLoadResource: (id)    sender;
{
    NSString *path = [[NSBundle mainBundle] pathForResource:@"file"
                                              ofType:@"txt"];

    NSError *error;
    NSString *str = [NSString stringWithContentsOfFile:path
                        encoding:NSUTF8StringEncoding error:&error];
    if (str != nil)
        m_cText.text = str;
            else
        m_cText.text = @"файл не найден";
}

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

- (void)viewDidUnload
{
}

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

@end
```

Рассмотрим приведенный код подробнее. Сами функции чтения и записи строки `writeToFile` и `stringWithContentsOfFile` принадлежат к классу `NSString`, они никаких трудностей не вызывают. Основная сложность состоит в получении корректного пути к файлу, для этого используется функция `NSSearchPathForDirectoriesInDomains` с параметром `NSDocumentDirectory`. Для получения пути к файлу в ресурсах программы мы используем функцию `NSBundle::pathForResource`.

Внешний вид программы показан на рис. 7.4.

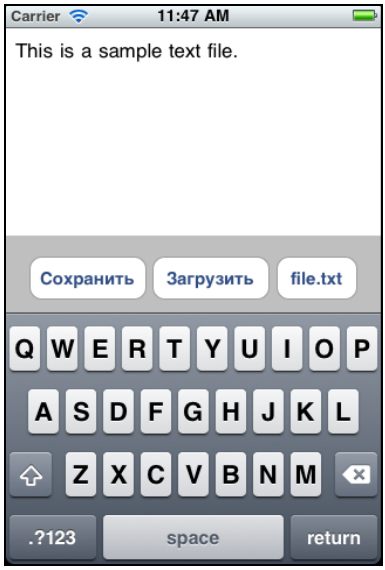


Рис. 7.4. Чтение и загрузка текстовых данных

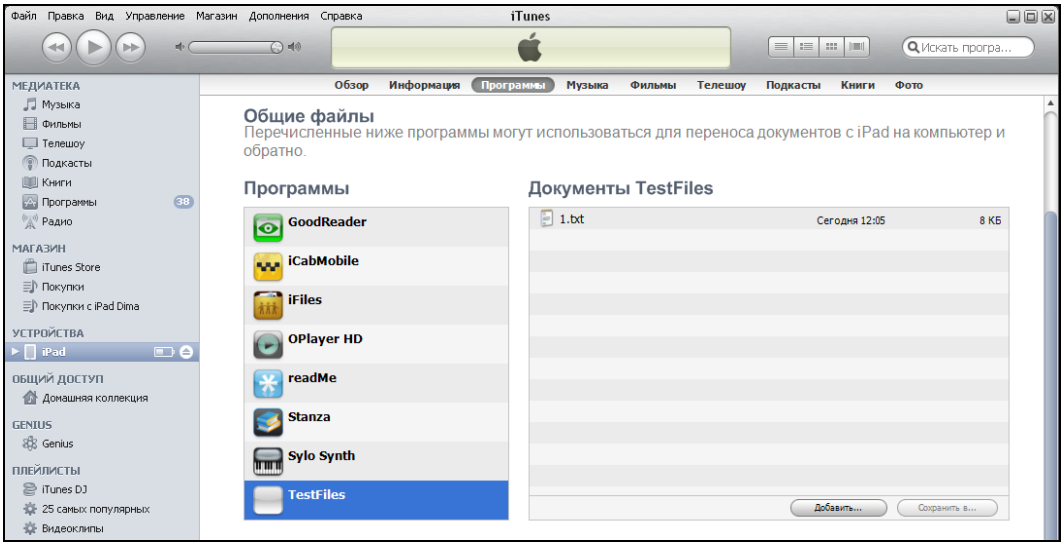


Рис. 7.5. Отображение файлов в iTunes

При нажатии на кнопку **file.txt** на экране должно появляться содержимое соответствующего файла, нажатие кнопок **Сохранить** и **Загрузить** должно приводить к выполнению указанных действий.

Важный момент — синхронизация данных с iTunes. Откроем файл TestFiles-Info.plist и добавим в него параметр `UIFileSharingEnabled` со значением `TRUE`. После этого нужно снова загрузить программу в смартфон, и если все было сделано

правильно, то после выполнения программы в iTunes появится новая строка с названием программы, как показано на рис. 7.5: в списке программ появилось наше приложение TestFiles, в списке общих файлов которого виден файл 1.txt.

Для чтения и записи файлов можно использовать не только высокоуровневые функции классов `NSString`, `UIImage`, но и стандартные С-функции `fopen`, `fread`, `fclose` и др. Пример чтения бинарных данных показан в листинге 7.6.

На этом мы закончим рассмотрение функций работы с файлами, более подробную документацию можно найти на сайте **http://developer.apple.com** и в документации по языку С.

Листинг 7.6. Чтение бинарного файла

```
NSString *path = [[NSBundle mainBundle] pathForResource:@"file"
                                                 ofType:@"bin"];
FILE *f = fopen([path cStringUsingEncoding:1], "rb");
if (f != NULL)
{
     while (1)
     {
          int data = 0;
          int size = fread(&data, sizeof(char), sizeof(data), f);
          if (size != sizeof(data)) break;

          // обработка данных…
     }
     fclose(f);
}
```

Сохранение и загрузка настроек программы

Используя мобильное устройство, владелец может в любой момент отвлечься на телефонный звонок. Поэтому хорошим тоном считается сохранение данных по окончании работы приложения. Например, удобно, если программа для просмотра словаря или чтения книг при следующем запуске открывается в том же самом месте, не заставляя листать текст с начала.

Конечно, чтение и сохранение данных различных типов возможно с помощью функций работы с файлами, которые рассматривались ранее. Но для облегчения работы в iOS имеются специальные функции, делающие чтение и запись более простыми. Рассмотрим пример реализации этих функций в iOS (листинг 7.7).

Листинг 7.7. Чтение и загрузка целочисленных значений

```
+ (void)        SetProfileInt: (int)nVal: (NSString*)sKey
{
       NSString *str = [NSString stringWithFormat:@"%d", nVal;
```

```
        // Сохранить
        [[NSUserDefaults standardUserDefaults] setObject:str
                                                forKey: sKey];
}

+ (NSNumber*) GetProfileInt: (NSString*)sKey
{
        NSString *str = [[NSUserDefaults standardUserDefaults]
                                                    objectForKey: sKey];
        if (str != nil && [str length] != 0)
        {
                int n_val = [str intValue];
                return [NSNumber numberWithInt:n_val];
        }
        return nil;
}
```

Мы используем функции класса `NSUserDefaults` для чтения и записи строковых констант, параллельно выполняя преобразование целых чисел в строку для сохранения. В классе `NSUserDefaults` помимо функции `setObject` имеются функции `setInteger` и `setFloat`, которые можно использовать для соответствующих типов данных.

Важно заметить, что операционная система iOS позволяет программисту не делать вообще какие-либо окна для изменения настроек программы. Для этого достаточно добавить в программу файл Root.plist, содержащий определенным образом сформированный XML-файл, и в окне настроек системы автоматически будет сформирована новая запись с соответствующим окном настроек. Файл Root.plist имеет достаточно сложную структуру, позволяющую хранить данные различных типов. Более подробное описание его можно найти в разделе "Implementing Application Preferences" на сайте Apple.

Запись и обработка звука

Запись звука является простой по сути, но довольно сложной из-за большого количества программного кода и обилия специфических функций. Поэтому мы не будем создавать программу "с нуля", а рассмотрим уже готовую, благо Apple предоставляет демонстрационный проект SpeakHere по адресу **developer.apple.com/library/ios/#samplecode/SpeakHere/Introduction/Intro.html**.

Для того чтобы использовать запись звука, нужно выполнить несколько действий:

1. Активировать звуковую подсистему с помощью функций `AudioSessionInitialize` и `AudioSessionSetActive`.
2. Заполнить параметры структуры `AudioStreamBasicDescription`, которая содержит такие поля как `mSampleRate`, `mBytesPerPacket`, `mChannelsPerFrame` и пр. Важно понимать назначение полей структуры `AudioStreamBasicDescription`, т. к. они отвечают за все параметры записи — битрейт, количество каналов и пр.

3. Создать новую "очередь записи" с помощью функции `AudioQueueNewInput` и добавить в нее буферы, предназначенные для хранения звука с помощью функции `AudioQueueAllocateBuffer`. Одним из параметров функции `AudioQueueNewInput` является адрес callback-функции, которая будет автоматически вызываться системой во время записи. Функция будет вызываться постоянно, как только очередной выделенный буфер заполняется записанными данными. Таким образом, частота вызовов зависит от объема выделенного буфера и битрейта, заданного во втором шаге. Необходимо выделить минимум два буфера: пока один обрабатывается, во второй автоматически происходит запись, благодаря этому процесс не прерывается, и в записи не будет щелчков и шумов.
4. Начать запись с помощью вызова функции `AudioQueueStart`.
5. В созданной нами callback-функции обеспечить нужную обработку приходящих звуковых блоков. В этой функции мы имеем указатель на "сырые" аудиоданные и можем делать с ними что угодно: сохранять в файл, выводить осциллограмму на экран и пр.

По завершении записи следует закрыть звуковое устройство с помощью вызова функций `AudioQueueStop` и `AudioQueueDispose`.

Рассмотрим теперь реализацию этого способа на примере программы SpeakHere. Демонстрационный проект состоит из нескольких файлов:

- `SpeakHereAppDelegate.h/m` — основной класс `ApplicationDelegate`, обеспечивающий запуск программы;
- `SpeakHereController.h/mm` — класс основного окна программы. Стоит обратить внимание на расширение mm, оно показывает, что в файле совместно используется Objective-C и C++;
- `AQRecorder.h/mm` — класс, отвечающий за запись звука;
- `AQPlayer.h/mm` — класс воспроизведения звука;
- `MeterTable.h/mm` — вспомогательный класс для подсчета количества "меток" вывода уровня в децибелах;
- `LevelMeter`, `LevelView`, `GLLevelMeter`, `AQLevelMeter` — вспомогательные классы для визуализации и вывода на экран.

При желании более детально разобраться с этими функциями можно скачать программу SpeakHere, скомпилировать и запустить на симуляторе или на реальном устройстве (рис. 7.6).

Существует еще один способ записи и воспроизведения звука — использование классов `AudioUnit`. Этот способ позволяет делать не только отдельно запись или воспроизведение, но опционально обеспечивает "сквозную" обработку звука, позволяя накладывать на записываемый (с микрофона) звук в реальном времени различные эффекты (например, эхо) и сразу же выводить его. Подсистема `AudioUnit` весьма обширна, она позволяет комбинировать несколько модулей (units) в граф обработки звука (audio processing graph), добавлять различные эффекты, например микширование. Подробное описание `AudioUnit` выходит за рамки этой книги, желающие могут обратиться к документации Apple.

Рис. 7.6. Запись и воспроизведение звука

На этом мы закончим рассмотрение функций взаимодействия с iOS и перейдем к не менее интересной теме — созданию игровых программ.

Глава 8

Основы создания игровых программ

Освоив основные приемы и принципы программирования для операционной системы iOS, мы перейдем к рассмотрению наиболее сложной категории программного обеспечения — игровых программ. Здесь есть два аспекта: во-первых, игры зачастую являются наиболее ресурсоемкими приложениями, во-вторых, они весьма требовательны к качеству графики и дизайна. Как минимум, желателен союз программиста и художника, а возможно и композитора, лишь тогда игра может рассчитывать на коммерческий успех. Впрочем, целью данной книги является все-таки программирование, а не маркетинг или управление персоналом, поэтому мы будем рассматривать лишь программную составляющую. Все остальное можно найти в специальной литературе. Итак, приступим.

Простая 2D-игра с использованием элементов управления iOS

Операционная система iOS предоставляет весьма развитые средства для работы с графикой и звуком. Это поддержка OpenGL ES 1.1 и 2.0, возможность использования шейдеров и специальных 2D-библиотек. Для создания несложных игр хватит встроенных средств графического интерфейса — изображения с прозрачностью, анимации, даже матричные преобразования. Это позволяет делать игры средствами стандартных компонентов интерфейса iOS, не прибегая к изучению OpenGL или других специализированных библиотек. Конечно, вряд ли таким способом удастся создать шутер или авиасимулятор, но "крестики-нолики" или "морской бой" вполне можно сделать.

Описание игровой логики заняло бы много времени, поэтому рассмотрим более простой пример. *В главе 5* рассматривалось движение ящика. Добавим возможность стрелять из пушки по ящику. Для этого создадим изображения пушки и снарядов, затем соединим все это вместе.

Вначале создадим изображение пушки в Adobe Photoshop с параметрами, как на рис. 8.1: **Color Mode** (Режим цвета) установлен в значение **RGB Color** и **Background Contents** (Фон) установлен в значение **Transparent** (Прозрачный). Изображение с прозрачным фоном нам нужно для того, чтобы пушка корректно рисовалась поверх других компонентов, в противном случае изображение будет иметь квадратные углы.

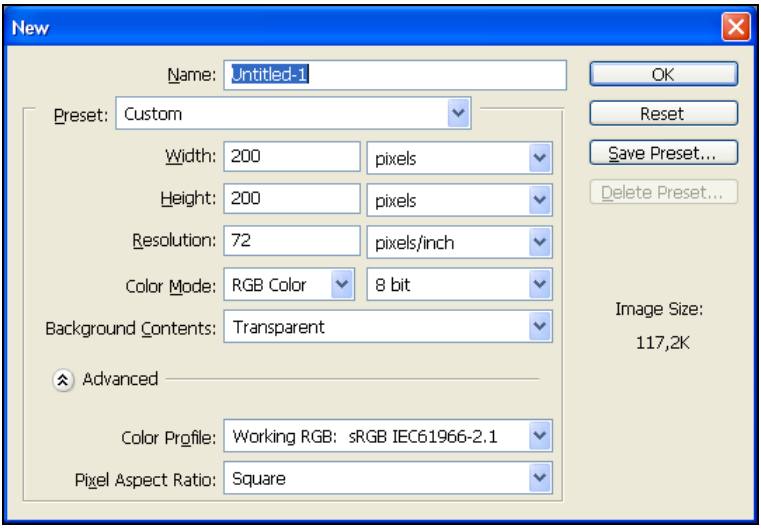


Рис. 8.1. Создание изображения в Adobe Photoshop

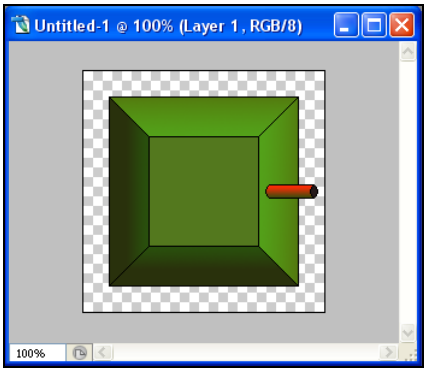


Рис. 8.2. Готовое изображение

Рис. 8.3. Кнопка стрельбы

Изображение должно выглядеть так, чтобы оно было симметрично относительно центра, это позволит вращать пушку вокруг центральной оси.

Конечно, вряд ли пушка на рис. 8.2 позволит выиграть конкурс на лучший дизайн, но для учебного примера ее вполне достаточно. Прозрачные области по краям показаны черно-белой сеткой. Нарисуем ядро размером 10×10 пикселов, кнопки **Fire** и **Restart**.

Нам необходимо сделать два изображения кнопки, одно для обычного, второе — для нажатого состояния. Также обратим внимание на прозрачную область вокруг кнопки (рис. 8.3). Изображение ящика, по которому будем стрелять, возьмем из рассмотренного ранее примера *главы 5*. Теперь, когда графика готова, можно приступать к программированию.

Разместим на главном окне изображение пушки и цели. Привяжем изображения к переменным класса `UIImageView`. Поскольку единственным элементом управле-

ния в iPhone является сенсорный экран, сделаем возможность поворачивать пушку нажатием пальца, для чего добавим соответствующие обработчики `touchesBegan`, `touchesMoved` и `touchesEnded`. У каждого объекта интерфейса есть свойство `transform`, изменяя которое мы можем задавать графическую трансформацию, в том числе и поворот объекта.

Теперь собственно процесс стрельбы. Пользователь может нажать кнопку **Fire** много раз, поэтому мы будем хранить массив объектов `UIImageView`. В отличие от остальных элементов интерфейса, которые мы создавали через Interface Builder, объект "ядро" мы будем создавать программно, в момент нажатия пользователем кнопки **Fire**. Поскольку пушка может быть повернута на разные углы, мы должны вместе с ядром хранить и направление его полета (очевидно, в момент выстрела направления дула пушки и ядра совпадают, но после выстрела ядро и пушка уже не связаны друг с другом). Для визуализации полета ядер их координаты будут пересчитываться по таймеру с шагом в 1/30 секунды. Если ядро попадает в цель или улетает за пределы экрана, соответствующий элемент массива `UIImageView` будет освобождаться.

Рассмотрим реализацию этих действий, как показано в листинге 8.1.

Листинг 8.1. Реализация стрельбы из пушки с помощью элементов GUI

Файл GameUIViewController.h:

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

@interface GameUIViewController : UIViewController
{
        IBOutlet UIImageView  *mGun;
        IBOutlet UIImageView  *mTarget;

        // Поворот пушки
        CGFloatmfGunAngle, mfStartAngle;
        BOOL   mbIsTouch;

        // Ядра
        #define MAX_COUNT 100
        UIImageView *pBullets[MAX_COUNT];
        float          fBulletsAngle[MAX_COUNT];

        NSTimer                        *pTimer;
        NSLock                         *pLock;
}

- (IBAction) onButtonFire:(id)sender;
- (IBAction) onButtonRestart:(id)sender;
- (CGFloat) AngleFromPt: (CGPoint)pt: (CGPoint)center;
- (void) doStep;
```

```
@property(retain, nonatomic) UIImageView  *mGun;
@property(retain, nonatomic) UIImageView  *mTarget;

@end
```

Файл GameUIViewController.m:

```
#import "GameUIViewController.h"

@implementation GameUIViewController
@synthesize mGun;
@synthesize mTarget;

// Implement viewDidLoad to do additional setup after loading the view,
// typically from a nib.
- (void)viewDidLoad
{
    [super viewDidLoad];

    mfGunAngle = 0;
    mbIsTouch = FALSE;

    pLock = [NSLock new];

    for(int p=0; p<MAX_COUNT; p++)
        pBullets[p] = nil;

    pTimer = [[NSTimer scheduledTimerWithTimeInterval:1.0/30.0
                      target:self selector:@selector(doStep)
                      userInfo:nil repeats:YES] retain];

}

- (void) doStep
{
    [pLock lock];

    // Движение ядер
    for(int p=0; p<MAX_COUNT; p++)
    {
        if (pBullets[p] == nil) continue;

        float dx = cos(fBulletsAngle[p]),
              dy = sin(fBulletsAngle[p]), speed = 2;
        float x = pBullets[p].center.x + speed*dx,
              y = pBullets[p].center.y + speed*dy;
```

```
        if (x < 0 || x >= self.view.bounds.size.width ||
            y < 0 || y >= self.view.bounds.size.height)
        {
                    [pBullets[p] removeFromSuperview];
                    [pBullets[p] release];
                    pBullets[p] = nil;

                    continue;
        }

        pBullets[p].center = CGPointMake(x, y);
    }

    // Проверка попадания
    if (mTarget.hidden != YES)
    {
        for(int p=0; p<MAX_COUNT; p++)
        {
            if (pBullets[p] == nil) continue;

            CGRect target = [self.view convertRect:[mTarget frame]
                                        fromView:self.view];
            if (CGRectContainsPoint(target, pBullets[p].center))
            {
                mTarget.hidden = YES;

                // Спрятать снаряд
                [pBullets[p] removeFromSuperview];
                [pBullets[p] release];
                pBullets[p] = nil;
                break;
            }
        }
    }

    [pLock unlock];
}

- (IBAction) onButtonRestart:(id)sender
{
    [pLock lock];

    // Показать цель снова
    mTarget.hidden = NO;
```

```
    for(int p=0; p<MAX_COUNT; p++)
    {
        if (pBullets[p] == nil) continue;

        // Убрать снаряды
        [pBullets[p] removeFromSuperview];
        [pBullets[p] release];
        pBullets[p] = nil;
    }

    mfGunAngle = 0;
    mGun.transform = CGAffineTransformMakeRotation(mfGunAngle);

    [pLock unlock];
}

- (IBAction) onButtonFire:(id)sender
{
    // Найти свободную ячейку
    for(int p=0; p<MAX_COUNT; p++)
    {
        if (pBullets[p] == nil)
        {
            float raduis = 105, img = 10,
                  x = mGun.center.x + raduis*cos(mfGunAngle),
                  y = mGun.center.y + raduis*sin(mfGunAngle);

            pBullets[p] = [[UIImageView alloc]
                   initWithFrame:CGRectMake(x-img/2, y-img/2, img, img)];
            [pBullets[p] setImage:[UIImage imageNamed:@"ImgBullet.png"]];
            fBulletsAngle[p] = mfGunAngle;

            [self.view addSubview:pBullets[p]];
            break;
        }
    }
}

// Override to allow orientations other than the
// default portrait orientation.
- (BOOL)shouldAutorotateToInterfaceOrientation:
                    (UIInterfaceOrientation)interfaceOrientation
{
    // Return YES for supported orientations
    return (interfaceOrientation == UIInterfaceOrientationPortrait ||
```

```
        interfaceOrientation == UIInterfaceOrientationPortraitUpsideDown);
}

- (void)touchesBegan:(NSSet *)touches withEvent:(UIEvent *)event
{
    UITouch *touch = [touches anyObject];
    CGPoint touchStart = [touch locationInView:self.view];

    CGRect rect = [self.view convertRect:[mGun frame]
                        fromView:self.view];
    if (CGRectContainsPoint(rect, touchStart))
    {
        mbIsTouch = TRUE;

        CGPoint center = CGPointMake(CGRectGetMidX(rect),
                                     CGRectGetMidY(rect));

        mfStartAngle = -mfGunAngle + [self AngleFromPt: touchStart:
                                                        center];
   }
}

- (void)touchesMoved:(NSSet *)touches withEvent:(UIEvent *)event
{
    if (mbIsTouch)
    {
        UITouch *touch = [touches anyObject];
        CGPoint touchStart = [touch locationInView:self.view];

        CGRect  rect = [self.view convertRect:[mGun frame]
                                fromView:self.view];
        CGPoint center = CGPointMake(CGRectGetMidX(rect),
                                     CGRectGetMidY(rect));

        mfGunAngle = [self AngleFromPt: touchStart: center] -
                                                    mfStartAngle;
        mGun.transform = CGAffineTransformMakeRotation(mfGunAngle);
    }
}


- (void)touchesEnded:(NSSet *)touches withEvent:(UIEvent *)event
{
    if (mbIsTouch)
        mbIsTouch = FALSE;
```

```
}

- (CGFloat) AngleFromPt: (CGPoint)pt: (CGPoint)center
{
    CGFloat dx = pt.x - center.x, dy = pt.y - center.y,
            len = sqrt(dx*dx + dy*dy);

    CGFloat angle = acos(dx/len);
    if (dy < 0) angle = -angle;

    return angle;
}

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

- (void)viewDidUnload
{
}

- (void)dealloc
{
     [mGun release];
     [mTarget release];

     [pTimer invalidate];
     [pTimer release];

     [pLock release];
     for(int p=0; p<MAX_COUNT; p++)
     {
        if (pBullets[p] == nil) continue;

        [pBullets[p] removeFromSuperview];
        [pBullets[p] release];
        pBullets[p] = nil;
     }

     [super dealloc];
}

@end
```

В функции `viewDidLoad` происходит начальная инициализация объектов и запуск таймера, который будет вызывать функцию `doStep` каждые 1/30 секунды. В функции `doStep` выполняются два действия: движение ядер и проверка их выхода за экран или столкновения с целью. Если ядро улетело или попало в цель, оно удаляется из окна с помощью функции `removeFromSuperview`, и данный элемент массива освобождается. Для движения ядра к его координатам прибавляется величина, пропорциональная направлению его полета, соответствующие приращения вычисляются как косинус и синус угла. Функция `onButtonRestart` выполняет примерно те же действия: удаляет все ядра из массива и устанавливает угол наклона пушки в начальное положение. Функция `onButtonFire` ищет свободный элемент массива ядер и добавляет новое ядро с помощью вызова функции `UIImageView:initWithFrame`, затем этот объект добавляется в окно с помощью вызова функции `addSubview`. Функции `touchesBegan`, `touchesMoved` и `touchesEnded` обеспечивают поворот пушки с помощью движения пальца. Общий вид игры примерно такой, как на рис. 8.4. Конечно, попасть по неподвижному ящику большого труда не составляет, в качестве дополнительного упражнения читатели могут сделать его движущимся. Для этого нужно лишь дописать код в функции `doStep`, алгоритм движения ящика читатели также могут придумать самостоятельно. Можно предположить движения ящика влево или вправо, хранить дополнительную переменную текущей "фазы" и изменять *X*-координату как косинус этой величины. Можно добавить звуковые эффекты стрельбы и попадания ядра в цель (как использовать звуки, рассказывалось в предыдущей главе).

Даже на простом примере можно видеть, что обычными средствами интерфейса iOS можно создавать интерактивные графические приложения.

Теперь мы рассмотрим, как сделать игры "приближенными к реальному миру" с помощью библиотеки физического моделирования Box2D.

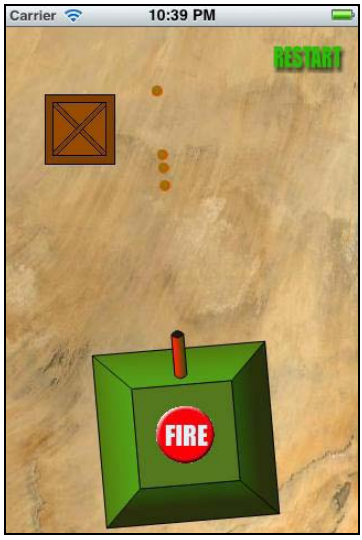


Рис. 8.4. Законченная версия игры "Попади в ящик"

Библиотека физического моделирования Box2D

Несмотря на большое распространение трехмерных игр, игры "на плоскости" имеют не меньшую популярность. В качестве яркого примера можно привести игру Angry Birds, которая несколько месяцев держалась в топах продаж App Store. Внешний вид игры показан на рис. 8.5.

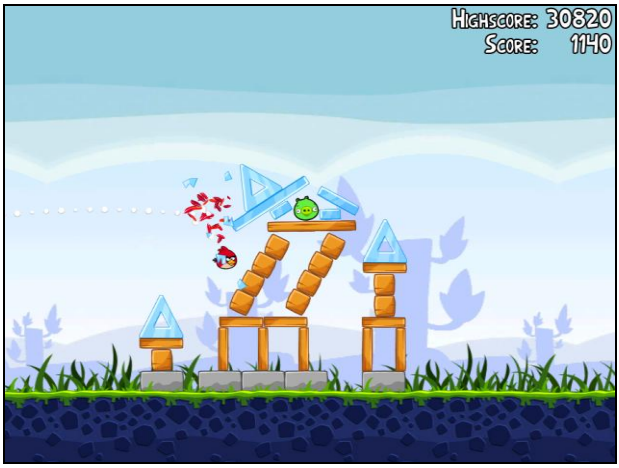


Рис. 8.5. Игра Angry Birds

Цель игры достаточно необычная — нужно из рогатки стрелять птицами по свиньям. Неизвестно, как авторов посетила идея со свиньями, но она оказалась весьма кстати: согласно опубликованным в сети Интернет данным количество платных скачиваний этой игры превысило 12 млн экземпляров.

Если еще раз посмотреть на рис. 8.5, то можно видеть типичный пример использования физического моделирования. Сцена состоит из некоторого количества статических (земля, камни) и динамических объектов (балки, бревна, птицы). В игре птица с некоторой скоростью ударяется в здание, что приводит к смещению объектов и падению всех или некоторых частей конструкции. Все динамические объекты имеют различные параметры: массу, форму, коэффициент трения, поэтому самостоятельно рассчитать движение объектов было бы не так просто. К счастью, эта проблема уже решена, существуют разнообразные библиотеки физического моделирования, специально предназначенные для решения подобных задач. Одну из них мы и рассмотрим.

Библиотека Box2D имеет в настоящее время большую популярность, благодаря следующим особенностям:

- это объектно-ориентированная библиотека, написанная на "чистом" C++, что позволило портировать ее на большое количество платформ, от iPhone и Windows Mobile до C# и Web-приложений, создаваемых на Flash;

- библиотека имеет большие возможности: расчет столкновений и движений тел, возможность подвижных или статичных соединений нескольких тел, возможность учета различных физических параметров, таких как масса, форма, коэффициент трения;
- библиотека распространяется бесплатно в исходных кодах, что позволяет легко подключить ее к любому приложению, лицензия разрешает ее использование в любых проектах, в том числе и коммерческих;
- в комплект поставки Box2D входит большое количество примеров и удобное приложение для тестирования.

Для начала необходимо скачать исходные файлы библиотеки с сайта **http://www.box2d.org** — и можно приступать к ее изучению.

Проект Hello World в Box2D

Создадим новый проект с названием Box2dHelloWorld. Для того чтобы иметь возможность подключить Box2D, вначале нужно переименовать файл Box2dHelloWorldViewController.m в Box2dHelloWorldViewController.mm. Это расширение указывает среде XCode, что в файле будет использоваться не только Objective-C, но и C++. Добавим к файлу строку `#import "Box2D/Box2D.h"` встроенных в Box2D типов данных. Теперь можно приступить к созданию виртуального физического мира.

Для физического моделирования следует выполнить несколько действий.

1. Создать объект "мир". Этот объект класса `b2World` является основной сущностью, создающей и хранящей все остальные данные:

```
b2Vec2 gravity(0.0f, 9.8f);
bool doSleep = true;
b2World *pWorld = new b2World(gravity, doSleep);
```

   Как нетрудно догадаться, вектор `gravity` указывает направление свободного падения. Да, в нашем мире будет действовать и сила тяжести! Впрочем, если нужно смоделировать невесомость, параметр `gravity` можно сделать и нулевым.

2. Указать границы мира, поскольку размеры в Box2D задаются в метрах. Выглядит весьма заманчивым сохранять значения в пикселах, но это приведет к большой погрешности (представьте моделирование падения бруска размерами 800×600 или 1024×768 метра). Поэтому надо предусмотреть пересчет системы координат, например, указать, что соотношение экранных объектов к физическим — 1:10 (или 1:16, как больше нравится). Возможно, деление на 16 будет выполняться быстрее, но мы будем использовать соотношение 1:10, как более наглядное. В этом случае размеры нашего виртуального мира в координатах экрана iPhone будут соответствовать области 32×46 м, что вполне нормально.

   Итак, чтобы указать границы нашего мира, используем следующий код:

```
b2BodyDef groundBodyDef;
groundBodyDef.position.Set(0,0);
b2Body* groundBody = pWorld->CreateBody(&groundBodyDef);

b2PolygonShape groundBox;
```

```
float w = 32.0, h = 46.0, top = 0;
// bottom
groundBox.SetAsEdge(b2Vec2(0,top), b2Vec2(w,top));
groundBody->CreateFixture(&groundBox,0);
// top
groundBox.SetAsEdge(b2Vec2(0,h), b2Vec2(w,h));
groundBody->CreateFixture(&groundBox,0);
// left
groundBox.SetAsEdge(b2Vec2(0,h), b2Vec2(0,top));
groundBody->CreateFixture(&groundBox,0);
// right
groundBox.SetAsEdge(b2Vec2(w,h), b2Vec2(w,top));
groundBody->CreateFixture(&groundBox,0);
```

Теперь наш "мир" окружен границей, в пределах которой будут находиться все объекты. Стоит уточнить еще один важный факт: библиотека Box2D является абстрактной, математической. Она не привязана к какой-либо платформе, графической библиотеке или параметрам устройства. Мы можем получить лишь данные об объекте, например, его положение и угол наклона, а вывести его на экран — уже наша задача, к которой Box2D никакого отношения не имеет.

3. Добавить в "мир" необходимые объекты. Как уже говорилось ранее, объекты могут быть статическими (неподвижными) и динамическими. Рассмотрим создание статического объекта прямоугольной формы:

```
b2BodyDef bodyDef;
bodyDef.type = b2_staticBody;
bodyDef.position.Set(5.0, 5.0);
bodyDef.userData = 0;
b2Body *pBody1 = pWorld->CreateBody(&bodyDef);

b2PolygonShape staticBox;
staticBox.SetAsBox(4.0, 4.0);
b2FixtureDef fixtureDef;
fixtureDef.shape = &staticBox;
pBody1->CreateFixture(&fixtureDef);
```

Мы указали тип объекта `b2_staticBody`, его положение `(5.0, 5.0)` и размер с помощью функции `SetAsBox`.

Динамический объект создается аналогичным образом:

```
b2BodyDef bodyDef;
bodyDef.type = b2_dynamicBody;
bodyDef.position.Set(1.0, 1.0);
bodyDef.userData = 0;
b2Body *pBody2 = pWorld->CreateBody(&bodyDef);

b2PolygonShape dynamicBox;
dynamicBox.SetAsBox(2.0, 1.0);
```

```
b2FixtureDef fixtureDef;
fixtureDef.shape = &dynamicBox;
pBody2->CreateFixture(&fixtureDef);
```

4. Активировать таймер, выполняющий пересчет объектов мира. После выполнения предыдущих шагов мы уже имеем готовый "мир", но он является статичным. Для того чтобы он "ожил", нужно запустить таймер, который будет обновлять содержимое "мира" каждые 1/30 секунды:

```
NSTimer *pTimer = [NSTimer scheduledTimerWithTimeInterval:1.0/30.0
                    target:self
                  selector:@selector(doStep)
                  userInfo:nil repeats:YES];
```

   Функция-обработчик события от таймера показана далее:

```
- (void) doStep
{
    float timeStep = 1.0f/30.0f;
    int32 velocityIterations = 8;
    int32 positionIterations = 1;
    pWorld->Step(timeStep, velocityIterations, positionIterations);

    [self updateImages];
}
```

   Функции `Step` передан в качестве первого параметра шаг временного интервала. Параметр `Iterations` отвечает за точность работы алгоритмов расчета: чем больше количество итераций, тем выше точность, но больше время вычислений.

   Box2D ничего "не знает" об экранном представлении, поэтому мы самостоятельно обновляем содержимое экрана после выполнения расчетов, вызывая функцию `updateImages`.

5. По завершении работы программы мы корректно убираем за собой все созданные объекты:

```
pWorld->DestroyBody(pBody1);
pWorld->DestroyBody(pBody2);
pWorld->DestroyBody(groundBody);

delete pWorld;
pWorld = NULL;
```

Несмотря на громоздкий код, все довольно просто. Рассмотрим практический пример: поместим в верхней части экрана ящик с надписью "Hello World", который будет падать вниз, встречая на своем пути несколько препятствий. Приведенный код несложно адаптировать для этой задачи. Добавим в основное окно три изображения — картинку с ящиком и тремя "прибитыми" брусками. Начальное положение объектов показано на рис. 8.6.

Размер ящика — 175×40 пикселов, размер брусков — 20×20 пикселов, в наших "виртуальных" координатах это 17,5×4,0 и 2×2 соответственно. Добавим в проект

кнопку **Старт** и четыре компонента `UIImageView`, как показано на рис. 8.6. Установим связи через Interface Builder обычным способом.

Мы имеем один динамический объект и три статических. Код программы показан в листинге 8.2.

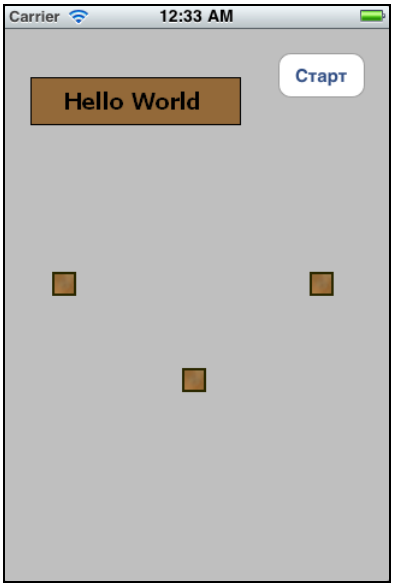


Рис. 8.6. Объекты в Box2D

Листинг 8.2. Проект с использованием Box2D

Файл Box2DHelloWorldViewController.h:

```
#import "Box2D/Box2D.h"

@interface Box2DHelloWorldViewController : UIViewController
{
        // GUI
        IBOutlet UIImageView  *mBox;
        IBOutlet UIImageView  *mStatic1, *mStatic2, *mStatic3;
        float fStartX, fStartY;

        b2World *pWorld;
        b2Body  *pBodyBox, *pBodyStatic1, *pBodyStatic2, *pBodyStatic3;

        NSTimer                         *pTimer;
}

- (void) updateImages;
```

```
- (void) createMainObject;
- (IBAction) onButtonRestart:(id)sender;

@property(retain, nonatomic) UIImageView  *mBox;
@property(retain, nonatomic) UIImageView  *mStatic1;
@property(retain, nonatomic) UIImageView  *mStatic2;
@property(retain, nonatomic) UIImageView  *mStatic3;
@end
```

Файл Box2DHelloWorldViewController.mm:

```
#import "Box2DHelloWorldViewController.h"

#define PTM_RATIO 10

@implementation Box2DHelloWorldViewController
@synthesize mBox;
@synthesize mStatic1;
@synthesize mStatic2;
@synthesize mStatic3;

// Implement viewDidLoad to do additional setup after loading the view,
// typically from a nib.
- (void)viewDidLoad
{
    [super viewDidLoad];

    b2Vec2 gravity(0.0f, 9.8f);
    bool doSleep = true;
    pWorld = new b2World(gravity, doSleep);

    //Create a ground box, the ground is defined as a rectangle.
    b2BodyDef groundBodyDef;
    groundBodyDef.position.Set(0, 0); //set with mid point, so /2
    b2Body* groundBody = pWorld->CreateBody(&groundBodyDef);

    b2PolygonShape groundBox;
    float w = 32.0, h = 46.0, top = 0;
    // bottom
    groundBox.SetAsEdge(b2Vec2(0,top), b2Vec2(w,top));
    groundBody->CreateFixture(&groundBox,0);
    // top
    groundBox.SetAsEdge(b2Vec2(0,h), b2Vec2(w,h));
    groundBody->CreateFixture(&groundBox,0);
    // left
    groundBox.SetAsEdge(b2Vec2(0,h), b2Vec2(0,top));
    groundBody->CreateFixture(&groundBox,0);
```

```
// right
groundBox.SetAsEdge(b2Vec2(w,h), b2Vec2(w,top));
groundBody->CreateFixture(&groundBox,0);

// Тело-1
b2BodyDef bodyDef;
bodyDef.type = b2_staticBody;
bodyDef.position.Set(mStatic1.center.x/PTM_RATIO,
                     mStatic1.center.y/PTM_RATIO);
bodyDef.userData = 0;
pBodyStatic1 = pWorld->CreateBody(&bodyDef);

b2PolygonShape staticBox;
staticBox.SetAsBox(0.5*mStatic1.frame.size.width/PTM_RATIO,
                   0.5*mStatic1.frame.size.height/PTM_RATIO);
b2FixtureDef fixtureDef;
fixtureDef.shape = &staticBox;
pBodyStatic1->CreateFixture(&fixtureDef);

// Тело-2
bodyDef.position.Set(mStatic2.center.x/PTM_RATIO,
                     mStatic2.center.y/PTM_RATIO);
pBodyStatic2 = pWorld->CreateBody(&bodyDef);
staticBox.SetAsBox(0.5*mStatic2.frame.size.width/PTM_RATIO,
                   0.5*mStatic2.frame.size.height/PTM_RATIO);
pBodyStatic2->CreateFixture(&fixtureDef);

// Тело-3
bodyDef.position.Set(mStatic3.center.x/PTM_RATIO,
                  mStatic3.center.y/PTM_RATIO);
pBodyStatic3 = pWorld->CreateBody(&bodyDef);
staticBox.SetAsBox(0.5*mStatic3.frame.size.width/PTM_RATIO,
                   0.5*mStatic3.frame.size.height/PTM_RATIO);
pBodyStatic3->CreateFixture(&fixtureDef);

// Позиция
fStartX = mBox.center.x/PTM_RATIO;
fStartY = mBox.center.y/PTM_RATIO;

pBodyBox = NULL;
[self createMainObject];

pTimer = [[NSTimer scheduledTimerWithTimeInterval:1.0/30.0
                   target:self selector:@selector(doStep)
                   userInfo:nil repeats:YES] retain];
```

```
}

- (void) createMainObject
{
    if (pBodyBox != NULL)
    {
        pWorld->DestroyBody(pBodyBox);
        pBodyBox = NULL;
    }

    mBox.center = CGPointMake(PTM_RATIO*fStartX, PTM_RATIO*fStartY);
    mBox.transform = CGAffineTransformMakeRotation(0);

    float     obj_w = mBox.frame.size.width,
              obj_h = mBox.frame.size.height;

    b2BodyDef bodyDef;
    bodyDef.type = b2_dynamicBody;
    bodyDef.position.Set(fStartX, fStartY);
    bodyDef.userData = 0;
    bodyDef.angle = 0;
    pBodyBox = pWorld->CreateBody(&bodyDef);

    b2PolygonShape dynamicBox;
    dynamicBox.SetAsBox(0.5*obj_w/PTM_RATIO, 0.5*obj_h/PTM_RATIO);

    b2FixtureDef fixtureDef;
    fixtureDef.shape = &dynamicBox;
    fixtureDef.density = 1.0f;
    fixtureDef.friction = 0.3f;
    fixtureDef.restitution = 0.3f;
    pBodyBox->CreateFixture(&fixtureDef);
}

- (void) doStep
{
    float timeStep = 1.0f/30.0f;
    int32 velocityIterations = 8;
    int32 positionIterations = 1;
    pWorld->Step(timeStep, velocityIterations, positionIterations);

    [self updateImages];
}

- (void) updateImages
```

```
{
    float x = pBodyBox->GetPosition().x, y = pBodyBox->GetPosition().y;
    mBox.center    = CGPointMake(PTM_RATIO*x, PTM_RATIO*y);
    mBox.transform = CGAffineTransformMakeRotation(pBodyBox->GetAngle());
}

- (IBAction) onButtonRestart:(id)sender
{
     [self createMainObject];
     [self updateImages];
}


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

- (void)viewDidUnload
{
}

- (void)dealloc
{
    [mStatic1 release];
    [mStatic2 release];
    [mStatic3 release];
    [mBox release];

    [pTimer invalidate];
    [pTimer release];

    pWorld->DestroyBody(pBodyBox);
    pWorld->DestroyBody(pBodyStatic1);
    pWorld->DestroyBody(pBodyStatic2);
    pWorld->DestroyBody(pBodyStatic3);

    delete pWorld;
    pWorld = NULL;

    [super dealloc];
}
@end
```

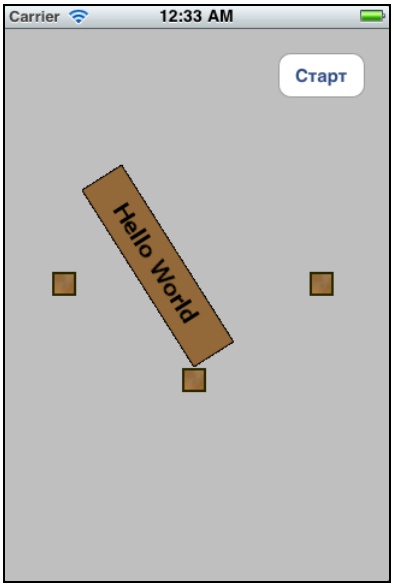


Рис. 8.7. Объекты в Box2D — работа программы

Всю "физическую" работу библиотека делает автоматически (рис. 8.7). Для того чтобы проект скомпилировался, необходимо добавить к нему исходные коды библиотеки Box2D из архива (в папке примеров можно взять проект Box2DHelloWorldSrc.zip, содержащий все необходимые файлы).

Управление объектами в Box2D

То, что мы сделали, уже немало. Однако объект, который просто падает вниз, лишен практического смысла. Необходимо управлять объектами приложением необходимых сил.

Листинг 8.3. Приложение силы к объекту

```
b2Vec2 f1 = pBody->GetWorldVector(b2Vec2(0, 80));
b2Vec2 p1 = pBody->GetWorldPoint(b2Vec2(0.0f, 4.0f));
pBody->ApplyForce(f1, p1);
```

Сила задается точкой приложения и вектором направления. В листинге 8.3 сила прикладывается к точке с координатами (0,4) относительно центра объекта. Вызов этой функции удобно вставить в функцию обработки таймера, прилагая силу в заданные моменты согласно сценарию программы. Параметрами функции `ApplyForce` являются значения векторов в мировых координатах. Это удобно для представления "глобальных" сил, например силы тяжести, однако для управления объектом используются функции `GetWorldVector` и `GetWorldPoint`, позволяющие преобразовывать "локальные" координаты в "мировые".

Рассмотрим практический пример — сделаем простую программу, позволяющую управлять космическим кораблем. У корабля будет два двигателя, нажатием на экран можно включать любой из них. Нижнюю часть экрана мы используем для управления двигателем, остальная область будет отведена для "полетной зоны".

Для создания программы выполним несколько шагов.

1. Создадим новый проект. Создадим спрайт корабля и привяжем его к элементу интерфейса `UIImageView`, создадим также две кнопки, которые будут управлять двигателями.
2. Создадим объект "мир" с помощью Box2D, настроим его границы и добавим одно динамическое тело — космический корабль.
3. Добавим две булевы переменные, отвечающие за левый и правый двигатели, обработчики нажатия и отпускания кнопок, устанавливающие значения соответствующих переменных.
4. В функции таймера осуществим обновление спрайта корабля и приложение к нему силы, соответствующей каждому из двигателей. В верхней части окна выведем скорость корабля.

Код программы в листинге 8.4 похож на предыдущий, поэтому показаны только основные его части.

Листинг 8.4. Программа управления космическим кораблем

Файл Box2DSpaceShipViewController.h:

```
#import <UIKit/UIKit.h>
#import "Box2D/Box2D.h"

@interface Box2DSpaceShipViewController : UIViewController
{
        // GUI
        IBOutlet UIImageView  *mShip;
        IBOutlet UILabel      *mLabelInfo;
        BOOL     bEngineL, bEngineR;

        // Box2D
        b2World *pWorld;
        b2Body  *pBodyShip;
        float fStartX, fStartY;

        NSTimer*pTimer;
}

@property(retain, nonatomic) UIImageView  *mShip;
@property(retain, nonatomic) UILabel      *mLabelInfo;

- (void) createMainObject;
```

```
- (void) updateImages;
- (void) doStep;
- (IBAction) onButtonEngineLDown:(id)sender;
- (IBAction) onButtonEngineLUp:(id)sender;
- (IBAction) onButtonEngineRDown:(id)sender;
- (IBAction) onButtonEngineRUp:(id)sender;
- (IBAction) onButtonRestart:(id)sender;

@end
```

Файл Box2DSpaceShipViewController.mm:

```
#import "Box2DSpaceShipViewController.h"

#define PTM_RATIO 10

@implementation Box2DSpaceShipViewController
@synthesize mShip;
@synthesize mLabelInfo;

// Implement viewDidLoad to do additional setup after loading the view,
// typically from a nib.
- (void)viewDidLoad
{
    [super viewDidLoad];

    b2Vec2 gravity(0.0f, 2.0f);
    bool doSleep = true;
    pWorld = new b2World(gravity, doSleep);

    //Create a ground box, the ground is defined as a rectangle.
    b2BodyDef groundBodyDef;
    groundBodyDef.position.Set(0, 0); //set with mid point, so /2
    b2Body* groundBody = pWorld->CreateBody(&groundBodyDef);

    b2PolygonShape groundBox;
    float w = 32.0, h = 31.0, top = 0;
    // bottom
    groundBox.SetAsEdge(b2Vec2(0,top), b2Vec2(w,top));
    groundBody->CreateFixture(&groundBox,0);
    // top
    groundBox.SetAsEdge(b2Vec2(0,h), b2Vec2(w,h));
    groundBody->CreateFixture(&groundBox,0);
    // left
    groundBox.SetAsEdge(b2Vec2(0,h), b2Vec2(0,top));
    groundBody->CreateFixture(&groundBox,0);
    // right
```

```
    groundBox.SetAsEdge(b2Vec2(w,h), b2Vec2(w,top));
    groundBody->CreateFixture(&groundBox,0);

    // Положение
    fStartX = mShip.center.x/PTM_RATIO;
    fStartY = mShip.center.y/PTM_RATIO;

    pBodyShip = NULL;
    [self createMainObject];
    bEngineL = bEngineR = FALSE;

    pTimer = [[NSTimer scheduledTimerWithTimeInterval:1.0/30.0
                      target:self selector:@selector(doStep)
                      userInfo:nil repeats:YES] retain];
}

- (void) createMainObject
{
    // Функция аналогична предыдущей
    …
}

- (void) doStep
{
    float timeStep = 1.0f/30.0f;
    int32 velocityIterations = 8;
    int32 positionIterations = 1;
    pWorld->Step(timeStep, velocityIterations, positionIterations);

    // Приложить силу - двигатели
    if (bEngineL)
    {
        b2Vec2 f2 = pBodyShip->GetWorldVector(b2Vec2(0, -200));
        b2Vec2 p2 = pBodyShip->GetWorldPoint(b2Vec2(-1.3f, 0.0f));
        pBodyShip->ApplyForce(f2, p2);
    }
    if (bEngineR)
    {
        b2Vec2 f2 = pBodyShip->GetWorldVector(b2Vec2(0, -200));
        b2Vec2 p2 = pBodyShip->GetWorldPoint(b2Vec2(1.3f, 0.0f));
        pBodyShip->ApplyForce(f2, p2);
    }

    // Обновить изображения
    [self updateImages];
```

```
    // Показать скорость
    float speed = pBodyShip->GetLinearVelocity().Length();
    mLabelInfo.text = [NSString stringWithFormat:@"Speed: %.1f", speed];
}

- (void) updateImages
{
    // Функция аналогична предыдущей
    …
}

- (IBAction) onButtonEngineLDown:(id)sender
{
       bEngineL = TRUE;
}

- (IBAction) onButtonEngineLUp:(id)sender
{
       bEngineL = FALSE;
}

- (IBAction) onButtonEngineRDown:(id)sender
{
       bEngineR = TRUE;
}

- (IBAction) onButtonEngineRUp:(id)sender
{
       bEngineR = FALSE;
}

- (IBAction) onButtonRestart:(id)sender
{
       [self createMainObject];
       [self updateImages];
}

- (void)dealloc
{
     [super dealloc];

     [mShip release];
     [mLabelInfo release];

     [pTimer invalidate];
```

```
    [pTimer release];

    pWorld->DestroyBody(pBodyShip);

    delete pWorld;
    pWorld = NULL;
}

@end
```

В зависимости от выбранного двигателя мы прикладываем силу либо к левой, либо к правой половине космического корабля. Размеры рабочей области уменьшены, из-за того что часть пространства отведена под кнопки. Гравитация также уменьшена для того, чтобы имитировать полет корабля на планете с меньшей силой тяжести.

Две кнопки в нижней части экрана на рис. 8.8 зарезервированы для добавления функций, которые будут рассмотрены далее.

Рис. 8.8. Управление космическим кораблем в Box2D

Программу можно усовершенствовать. Например, ввести ограничение на количество топлива, для чего достаточно добавить в функцию таймера счетчик времени включения двигателей.

У приведенной модели космического корабля есть один недостаток. Как могли заметить внимательные читатели, для создания тела корабля используется функция `SetAsBox`, так что физическая модель корабля представляет собой прямоугольник.

Это может привести к не совсем точному поведению при угловом соударении корабля с землей. Как исправить эту проблему, будет рассказано при рассмотрении тел произвольной формы.

С помощью приложения сил легко варьировать поведение объекта: например, для взлета корабля с планеты можно имитировать сопротивление воздуха, для этого достаточно добавить силу, направленную против движения объекта:

```
b2Vec2 f1 = pBody->GetLinearVelocity();
b2Vec2 p1 = pBody->GetWorldPoint(b2Vec2(0, 0));
pBody->ApplyForce(-f1, p1);
```

Стоит обратить внимание на знак "минус" перед вектором силы. Теперь после выключения двигателей корабль будет довольно-таки быстро останавливаться.

Сопротивление может быть не только линейное, но и угловое. Данный код позволит останавливать вращение объекта:

```
pBody->ApplyAngularImpulse(-pBody->GetAngularVelocity());
```

Стрельба

Библиотека Box2D использует приближенные методы для вычисления физического взаимодействия, поэтому может существовать определенная погрешность расчетов. И если для больших объектов, таких как космический корабль, это незаметно, то для маленького и быстрого объекта (пуля или снаряд) это может стать критически важным. Никого не устроит, если быстро летящая пуля будет пролетать сквозь противника, не причиняя ему никакого вреда. Поэтому для объектов Box2D может использоваться специальный режим, который так и называется — "пуля" (bullet).

Листинг 8.5. Создание объекта типа "пуля"

```
b2BodyDef bodyDef;
bodyDef.type = b2_dynamicBody;
bodyDef.bullet = TRUE;
bodyDef.position.Set(1.0, 1.0);
bodyDef.userData = 0;
b2Body *pBullet = pWorld->CreateBody(&bodyDef);

b2PolygonShape dynamicBox;
dynamicBox.SetAsBox(2.0, 1.0);
b2FixtureDef fixtureDef;
fixtureDef.shape = &dynamicBox;
pBullets->CreateFixture(&fixtureDef);
pBullet->SetTransform(bodyDef.position, rotation);
```

В листинге 8.5 в третьей строке сверху параметр `bullet` устанавливается в `TRUE`. Последняя строка `SetTransform` не является обязательной, она показывает возможность задания угла направления — дула стреляющего объекта. Для того чтобы узнать направление, достаточно вызвать функцию `GetAngle`:

```
float rotation = pBody->GetAngle();
```

Аналогично можно узнать положение:

```
b2Vec2 pt = pBody->GetPosition();
```

Добавим две кнопки ведения огня: одна для пули, вторая — для бомбы (о ней мы поговорим позже), а также пару динамических объектов. Как и в листинге 8.1, будет создан массив объектов, хранящий "тела" пуль и их изображения. Обновленный код показан в листинге 8.6.

Листинг 8.6. Добавление стрельбы в программу

Файл Box2DSpaceShipViewController.h:

```
// Bullets
#define MAX_COUNT 100
UIImageView *pBullets[MAX_COUNT];
b2Body  *pBodyTarget1, *pBodyTarget2;
b2Body  *pBodyBullets[MAX_COUNT];
```

Файл Box2DSpaceShipViewController.mm:

```
- (void) updateImages
{
   float x = pBodyShip->GetPosition().x, y = pBodyShip->GetPosition().y;
   mShip.center=CGPointMake(PTM_RATIO*x, PTM_RATIO*y);
   mShip.transform=CGAffineTransformMakeRotation(pBodyShip->GetAngle());

   mTarget1.center= CGPointMake(PTM_RATIO*pBodyTarget1->GetPosition().x,
                                PTM_RATIO*pBodyTarget1->GetPosition().y);
   mTarget1.transform = CGAffineTransformMakeRotation(pBodyTarget1->
                                                      GetAngle());

   mTarget2.center = CGPointMake(PTM_RATIO*pBodyTarget2->
                                          GetPosition().x,
                                 PTM_RATIO*pBodyTarget2->
                                          GetPosition().y);
   mTarget2.transform = CGAffineTransformMakeRotation(pBodyTarget2->
                                                      GetAngle());

   for(int p=0; p<MAX_COUNT; p++)
   {
      if (pBullets[p] != nil && pBodyBullets[p] != NULL)
      {
         pBullets[p].center = CGPointMake(PTM_RATIO*pBodyBullets[p]->
                                                      GetPosition().x,
                                          PTM_RATIO*pBodyBullets[p]->
                                                      GetPosition().y);
         pBullets[p].transform =
             CGAffineTransformMakeRotation(pBodyBullets[p]->GetAngle());
      }
```

```
    }
}

- (IBAction) onButtonFire:(id)sender
{
    // find free place
    for(int p=0; p<MAX_COUNT; p++)
    {
        if (pBullets[p] == nil)
        {
            UIImage *bullet = [UIImage imageNamed:@"ImgBullet.png"],
                    *ship = [UIImage imageNamed:@"ImgShip.png"];

            float angle = pBodyShip->GetAngle() - M_PI/2;
            float x = mShip.center.x + ship.size.height*cos(angle);
            float y = mShip.center.y + ship.size.height*sin(angle);

            // Create image
            pBullets[p] = [[UIImageView alloc]
                           initWithFrame:CGRectMake(x-bullet.size.width/2,
                                                    y-bullet.size.height/2,
                                                    bullet.size.width,
                                                    bullet.size.height)];
            [pBullets[p] setImage:bullet];
            [self.view addSubview:pBullets[p]];

            // Создать снаряд
            b2BodyDef bodyDef;
            bodyDef.type = b2_dynamicBody;
            bodyDef.bullet = TRUE;
            bodyDef.position.Set(x/PTM_RATIO, y/PTM_RATIO);
            bodyDef.userData = 0;
            pBodyBullets[p] = pWorld->CreateBody(&bodyDef);

            b2PolygonShape dynamicBox;
            dynamicBox.SetAsBox(0.5*bullet.size.width/PTM_RATIO,
                                0.5*bullet.size.height/PTM_RATIO);
            b2FixtureDef fixtureDef;
            fixtureDef.restitution = 0.8;
            fixtureDef.shape = &dynamicBox;
            pBodyBullets[p]->CreateFixture(&fixtureDef);

            // Приложить силу
            b2Vec2 f  = pBodyShip->GetWorldVector(b2Vec2(0, -400));
            b2Vec2 pt = pBodyShip->GetWorldPoint(b2Vec2(0, 0));
```

```
            pBodyBullets[p]->ApplyForce(f, pt);

            // Отдача
            pBodyShip->ApplyForce(-f, pt);
            break;
        }
    }
}
```

По нажатию кнопки в функции `onButtonFire` создается новый объект "пуля", который инициализируется координатами стреляющего объекта. Одиночный импульс, имитирующий собственно выстрел, выполняется с помощью функции `ApplyForce`. С помощью этой же функции имитируется и отдача, для чего к основному объекту прикладывается такая же сила, действующая в противоположном направлении. В качестве объектов, по которым можно стрелять, выступают изображения ящиков из предыдущего примера (рис. 8.9).

Рис. 8.9. Возможность стрельбы в Box2D

Есть важный момент, требующий от нас углубленного знания C++. Как можно видеть, пули свободно летают по экрану, не причиняя никакого вреда. В настоящей игре такого, естественно, не должно быть. Для этого в Box2D существует возможность отслеживания столкновений между объектами.

Для того чтобы активировать ее, нужно сделать следующее.

1. Создать класс, наследуемый от класса `b2ContactListener`. Он содержит четыре виртуальные функции, которые можно переопределить. Целесообразно

запоминать индексы сталкивающихся объектов в массиве или каким-либо другим образом отмечать столкновения. Функция вызывается неоднократно во время выполнения алгоритмов Box2D, поэтому изменять в ней сами объекты (например, удалять их) нельзя. Можно лишь сохранять данные о столкновениях, чтобы использовать позднее.

2. Создать экземпляр этого класса и передать в Box2D указатель на него с помощью функции `pWorld->SetContactListener`.
3. В функции таймера после вызова функции `DoStep` следует проверить, не было ли столкновений, анализируя данные, сохраненные на первом шаге.

Рассмотрим пример класса в листинге 8.7.

Листинг 8.7. Класс, обрабатывающий столкновения

```
// Типы тел
#define BODY_GROUND 1
#define BODY_BULLET 2
#define BODY_BULLET_BLOW 20
#define BODY_TARGET 3
#define BODY_SHIP   4

class MyContactListener : public b2ContactListener
{
public:
    MyContactListener()
    {
    }

    virtual ~MyContactListener()
    {
    }

    virtual void EndContact(b2Contact* contact)
    {
        b2Fixture* fixtureA = contact->GetFixtureA();
        b2Fixture* fixtureB = contact->GetFixtureB();
        if (fixtureA != NULL && fixtureB != NULL)
        {
            b2Body *b1 = fixtureA->GetBody(),
                   *b2 = fixtureB->GetBody();
            if (b1 != NULL && b2 != NULL)
            {
                int t1 = (int)b1->GetUserData(),
                    t2 = (int)b2->GetUserData();
                if (t1 == BODY_BULLET && t2 == BODY_TARGET)
                     b1->SetUserData((void*)BODY_BULLET_BLOW);
```

```
                if (t1 == BODY_TARGET && t2 == BODY_BULLET)
                    b2->SetUserData((void*)BODY_BULLET_BLOW);
            }
        }
    }
};
```

В функции `EndContact` мы анализируем типы столкнувшихся объектов с помощью функции `GetUserData`. В нашем примере пуля может взрываться только при попадании в цель, поэтому если два типа тел соответствуют этому условию, значение `UserData` для пули изменяется. Впрочем, логика тут может быть любой, в зависимости от поставленной задачи. Если при создании тел поле `UserData` не заполнить, алгоритм работать не будет.

В функции `DoStep` выполняется проверка, если тип пули соответствует константе `BODY_BULLET_BLOW`, она удаляется:

```
for(int p=0; p<MAX_COUNT; p++)
{
    if (pBodyBullets[p] != nil &&
        (int)pBodyBullets[p]->GetUserData() == BODY_BULLET_BLOW)
    {
        // Удаление объекта, можно нарисовать взрыв в этом месте
        …
    }
}
```

Перед удалением мы можем сохранить в отдельной переменной его координаты, чтобы вывести спрайт взрыва в этом месте, например в виде объекта `UIImageView`. Кстати говоря, класс `UIImageView` имеет встроенный режим анимации, поэтому взрыв можно вполне натурально изобразить. С помощью этого же принципа можно выводить звуки столкновений при соударении различных предметов. Также имеется возможность оценить силу удара и тем самым изменять "живучесть" объекта при попадании в него.

Соединение нескольких тел

Создадим новый динамический объект "бомба". Для простоты мы сделаем бомбу прямоугольной.

```
b2Body * pBomb = pWorld->CreateBody(&bodyDef);
```

Вначале бомба должна быть жестко прикреплена к кораблю:

```
b2DistanceJointDef joint1;
joint1.bodyA = pBodyBomb;
joint1.bodyB = pBodyShip;
joint1.localAnchorA.Set(0,0);
joint1.localAnchorB.Set(0,0);
joint1.collideConnected = true;
b2Vec2 d = pBodyBomb->GetWorldPoint(joint1.localAnchorA) –
           pBodyShip->GetWorldPoint(joint1.localAnchorB);
joint1.length = d.Length();
b2Joint *pJoint = pWorld->CreateJoint(&joint1);
```

Как можно видеть, два объекта `pBodyBomb` и `pBodyShip` теперь соединены вместе, и само соединение сохраняется в виде отдельного объекта типа `b2Joint`. Этот объект нам пригодится, т. к. чтобы сбросить бомбу, соединение надо разорвать.

```
pWorld->DestroyJoint(pJoint);
pJoint = NULL;
```

Теперь траектории корабля и бомбы уже разные. Добавим код в нашу программу. Измененные фрагменты кода показаны в листинге 8.8.

Листинг 8.8. Возможность сбрасывания бомбы с корабля

Файл Box2DSpaceShipViewController.h:

```
@interface Box2DSpaceShipViewController : UIViewController
{
        // GUI
        IBOutlet UIImageView  *mShip, *mBomb, *mTarget1, *mTarget2;
        // Box2D
        b2World *pWorld;
        b2Body  *pBodyShip, *pBodyBomb, *pBodyTarget1, *pBodyTarget2;
        b2Joint *pJoint;
}

@property(retain, nonatomic) UIImageView  *mShip, *mBomb;

- (IBAction) onButtonBomb:(id)sender;
@end
```

Файл Box2DSpaceShipViewController.mm:

```
#import "Box2DSpaceShipViewController.h"

- (void) createMainObject
{
        if (pJoint != NULL)
        {
                pWorld->DestroyJoint(pJoint);
                pJoint = NULL;
        }
        if (pBodyShip != NULL)
        {
                pWorld->DestroyBody(pBodyShip);
                pBodyShip = NULL;
        }
        if (pBodyBomb != NULL)
        {
                pWorld->DestroyBody(pBodyBomb);
                pBodyBomb = NULL;
        }
```

```
        // Корабль
        pBodyShip = pWorld->CreateBody(&bodyDef);
        …

        // Бомба
        mBomb.transform = CGAffineTransformMakeRotation(0);
        bodyDef.position.Set(bodyDef.position.x, bodyDef.position.y + 2);
        bodyDef.userData = (void*)BODY_SHIP;
        pBodyBomb = pWorld->CreateBody(&bodyDef);
        …

        // Соединение
        b2DistanceJointDef joint1;
        joint1.bodyA = pBodyBomb;
        joint1.bodyB = pBodyShip;
        joint1.localAnchorA.Set(0,0);
        joint1.localAnchorB.Set(0,0);
        joint1.collideConnected = true;
        b2Vec2 d = pBodyBomb->GetWorldPoint(joint1.localAnchorA) —
                   pBodyShip->GetWorldPoint(joint1.localAnchorB);
        joint1.length = d.Length();
        pJoint = pWorld->CreateJoint(&joint1);

        // Цель-1
        …
}

- (void) updateImages
{
  float x = pBodyShip->GetPosition().x,
        y = pBodyShip->GetPosition().y;
  mShip.center = CGPointMake(PTM_RATIO*x, PTM_RATIO*y);
  mShip.transform = CGAffineTransformMakeRotation(pBodyShip->GetAngle());

  mBomb.center = CGPointMake(PTM_RATIO*pBodyBomb->GetPosition().x,
                             PTM_RATIO*pBodyBomb->GetPosition().y);
  mBomb.transform = CGAffineTransformMakeRotation(pBodyBomb->GetAngle());
  …
}

- (IBAction) onButtonBomb:(id)sender
{
        if (pJoint != NULL)
        {
                pWorld->DestroyJoint(pJoint);
                pJoint = NULL;
```

```
        }
}

- (void)dealloc
{
     [super dealloc];

     [mShip release];
     [mBomb release];
     [mTarget1 release];
     [mTarget2 release];
     [mLabelInfo release];

     [pTimer invalidate];
     [pTimer release];

     pWorld->DestroyBody(pBodyShip);
     pWorld->DestroyBody(pBodyBomb);
     pWorld->DestroyBody(pBodyTarget1);
     pWorld->DestroyBody(pBodyTarget2);
     if (pJoint != NULL)
         pWorld->DestroyJoint(pJoint);
     …
}
```

При сбросе бомбы кнопкой **Bomb** она должна отделяться от корабля (рис. 8.10).

Рис. 8.10. Сбрасывание бомбы

Создание тел произвольной формы

До этого времени мы рассматривали лишь прямоугольники, а в реальности объекты имеют более сложную форму. Для создания объекта он может задаваться в виде набора точек. Рассмотрим отличия старого и нового кода в листинге 8.9.

Листинг 8.9. Создание объекта

```
// Прямоугольный объект
b2PolygonShape dynamicBox;
dynamicBox.SetAsBox(2.0, 1.0);

// Объект сложной формы
b2PolygonShape dynamicBox;
//   5 *------------* 4
//    /              \
// 0 *                * 3
//    \              /
//   1 *------------* 2
float w = 2.9f, h = 0.6f, dw = 0.2f;
b2Vec2 vertices[6];
vertices[0].Set(-w-dw, 0);
vertices[1].Set(-w, -h);
vertices[2].Set( w, -h);
vertices[3].Set(w+dw, 0);
vertices[4].Set( w, h);
vertices[5].Set(-w, h);
dynamicBox.Set(vertices, 6);
```

Имеется два ограничения: количество вершин не должно превышать 8 (параметр хранится в виде константы `b2_maxPolygonVertices`) и, что более важно, фигура должна быть выпуклой (выпуклой считается та фигура, у которой все отрезки, соединяющие любые две точки, лежат внутри этой фигуры). Если по условию задачи фигура не такая, нужно разбить объект сложной формы на несколько выпуклых примитивов.

Для визуализации всех параметров Box2D можно использовать встроенную тестовую программу Testbed (рис. 8.11). Она позволяет не только наблюдать созданные объекты, но и управлять ими с помощью "мыши" или клавиатуры. Достаточно понятный исходный код позволяет легко изменять все необходимые параметры.

Полное описание функций библиотеки Box2D можно найти в сети Интернет, а также самостоятельно изучая исходные тексты примеров программы Testbed.

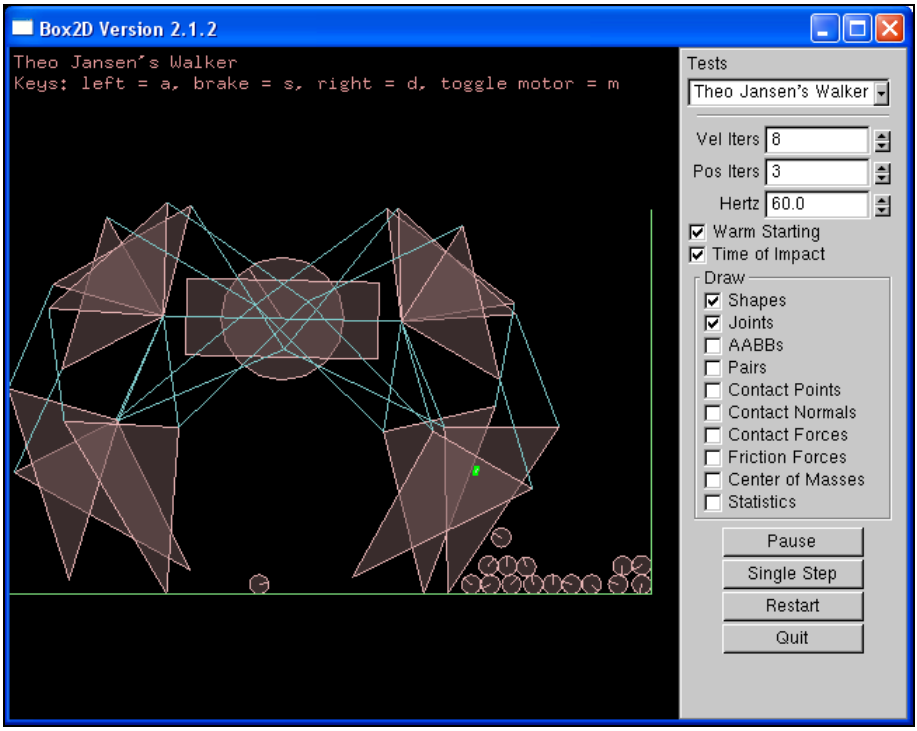


Рис. 8.11. Программа Testbed

Графическая библиотека Cocos2D

Предыдущие программы использовали стандартные средства графического интерфейса. Такой подход годится для простых приложений, однако для более сложных задач ресурсов устройства будет недостаточно. Единственной графической библиотекой, аппаратно поддерживаемой в операционной системе iOS, является OpenGL.

Программирование на низком уровне для начинающего может показаться сложным. Приходится отказаться от привычных элементов интерфейса и начинать "мыслить" низкоуровневыми примитивами — текстурами, вершинами, треугольниками и пр. Даже курсор мыши приходится выводить самостоятельно, не говоря уже о сложных объектах. Но, к счастью, эти проблемы возникли не вчера и фактически уже решены: существуют высокоуровневые библиотеки, способные взять большую часть работы на себя. Мы рассмотрим одну из популярных библиотек для iPhone — Cocos2D. Скачать последнюю версию библиотеки можно на сайте **http://www.cocos2d-iphone.org**. Популярность этой библиотеки обусловлена следующими факторами:

- бесплатность — лицензия позволяет использовать библиотеку в коммерческих приложениях без какой-либо оплаты;
- открытые исходные коды — библиотека поставляется полностью в исходных кодах и может легко подключаться к проекту;
- большое количество примеров.

Некоторые примеры использования можно найти по адресу **http://code.google.com/p/cocos2d-iphone/wiki/GamesUsingCocos2d**.

С помощью Cocos2D можно реализовать следующие действия:

- осуществлять вывод разнообразных графических примитивов с поддержкой различных анимационных эффектов (вращение, затенение и пр.);
- выводить текстовую информацию;
- выводить элементы графического интерфейса (кнопки, элементы меню) и получать сообщения от них;
- использовать иерархическое представление игровых экранов;
- комбинировать функции Cocos2D и непосредственно OpenGL в местах, требующих наибольшей производительности.

Библиотека Cocos2D содержит средства вывода звука и интегрированную библиотеку Box2D. Как и в случае Box2D, библиотека Cocos2D имеет принципиальное ограничение, она является двумерной. Поэтому желающим делать трехмерную игру придется обратиться к изучению OpenGL.

Проект Hello World в Cocos2D

По традиции начнем с проекта Hello World. Библиотека Cocos2D требует подключения довольно большого количества файлов, поэтому самым простым и удобным способом создания нового проекта является использование визарда, интегрирующегося в среду разработки XCode.

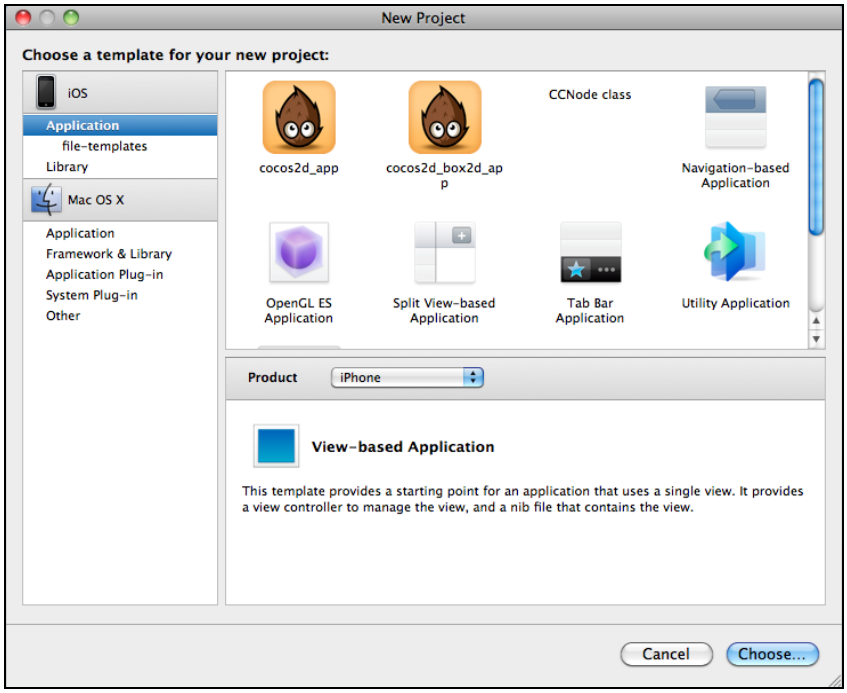


Рис. 8.12. Создание нового проекта Cocos2D

Для этого нужно распаковать папку cocos2d-iphone-0.99.5, найти в ней раздел templates и установить файлы из него в папку XCode. После этого в меню XCode при создании нового проекта должны появиться дополнительные пункты **cocos2d** и **cocos2d_box2d**, как показано на рис. 8.12.

Выберем первый пункт и рассмотрим подробнее файлы, созданные мастером (листинг 8.10).

Листинг 8.10. Файлы проекта Cocos2D — Cocos2DHelloWorldAppDelegate

```
#import "Cocos2DHelloWorldAppDelegate.h"
#import "cocos2d.h"
#import "HelloWorldScene.h"

@implementation Cocos2DHelloWorldAppDelegate

@synthesize window;

- (void) applicationDidFinishLaunching:(UIApplication*)application
{
        // CC_DIRECTOR_INIT()
        //
        // 1. Initializes an EAGLView with 0-bit depth format,
        // and RGB565 render buffer
        // 2. EAGLView multiple touches: disabled
        // 3. creates a UIWindow, and assign it to the "window" var
        // 4. Parents EAGLView to the newly created window
        // 5. Creates Display Link Director
        // 5a. If it fails, it will use an NSTimer director
        // 6. It will try to run at 60 FPS
        // 7. Display FPS: NO
        // 8. Device orientation: Portrait
        // 9. Connects the director to the EAGLView
        //
        CC_DIRECTOR_INIT();

        // Obtain the shared director in order to...
        CCDirector *director = [CCDirector sharedDirector];

        // Sets landscape mode
        [director setDeviceOrientation:kCCDeviceOrientationLandscapeLeft];

        // Turn on display FPS
        [director setDisplayFPS:YES];

        // Turn on multiple touches
        EAGLView *view = [director openGLView];
```

```
        [view setMultipleTouchEnabled:YES];

        // Default texture format for PNG/BMP/TIFF/JPEG/GIF images
        // It can be RGBA8888, RGBA4444, RGB5_A1, RGB565
        // You can change anytime.
        [CCTexture2D
           setDefaultAlphaPixelFormat:kTexture2DPixelFormat_RGBA8888];

        [[CCDirector sharedDirector] runWithScene: [HelloWorld scene]];
}


- (void)applicationWillResignActive:(UIApplication *)application {
        [[CCDirector sharedDirector] pause];
}

- (void)applicationDidBecomeActive:(UIApplication *)application {
        [[CCDirector sharedDirector] resume];
}

- (void)applicationDidReceiveMemoryWarning:(UIApplication *)application {
        [[CCDirector sharedDirector] purgeCachedData];
}

-(void) applicationDidEnterBackground:(UIApplication*)application {
        [[CCDirector sharedDirector] stopAnimation];
}

-(void) applicationWillEnterForeground:(UIApplication*)application {
        [[CCDirector sharedDirector] startAnimation];
}

- (void)applicationWillTerminate:(UIApplication *)application {
        [[CCDirector sharedDirector] end];
}

- (void)applicationSignificantTimeChange:(UIApplication *)application {
        [[CCDirector sharedDirector] setNextDeltaTimeZero:YES];
}

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

@end
```

В функции `applicationDidFinishLaunching` происходит инициализация библиотеки. Стоит обратить внимание на три функции:

- `setDisplayFPS` — включает отображение частоты кадров на экране, это полезно для отладочной версии;
- `setDeviceOrientation` — устанавливает ориентацию устройства. По умолчанию используется горизонтальный режим;
- `CCDirector::runWithScene` активизирует первую (и единственную в этом проекте) "сцену". Как уже говорилось, при работе с низкоуровневой графикой уже нет стандартного интерфейса, окон и диалогов. Поэтому для создания иерархии окон в Cocos2D используется понятие сцены. К ней мы сейчас и перейдем (листинг 8.11).

Листинг 8.11. Файлы проекта Cocos2D — HelloWorldScene

Файл HelloWorldScene.h:

```
#import "cocos2d.h"

// HelloWorld Layer
@interface HelloWorld : CCLayer
{
}
// returns a Scene that contains the HelloWorld as the only child
+(id) scene;

@end
```

Файл HelloWorldScene.m:

```
#import "HelloWorldScene.h"

// HelloWorld implementation
@implementation HelloWorld

+ (id) scene
{
    // 'scene' is an autorelease object.
    CCScene *scene = [CCScene node];

    // 'layer' is an autorelease object.
    HelloWorld *layer = [HelloWorld node];

    // add layer as a child to scene
    [scene addChild: layer];

    // return the scene
    return scene;
```

```
}

// on "init" you need to initialize your instance
- (id) init
{
    // always call "super" init
    // Apple recommends to re-assign "self" with the "super" return value
    if( (self=[super init] ))
    {
        // create and initialize a Label
        CCLabel* label = [CCLabel labelWithString:@"Hello World"
                                  fontName:@"Marker Felt" fontSize:64];

        // ask director the the window size
        CGSize size = [[CCDirector sharedDirector] winSize];

        // position the label on the center of the screen
        label.position =  ccp(size.width/2 , size.height/2);

        // add the label as a child to this Layer
        [self addChild: label];
    }
    return self;
}

// on "dealloc" you need to release all your retained objects
- (void) dealloc
{
    // in case you have something to dealloc, do it in this method
    // in this particular example nothing needs to be released.
    // cocos2d will automatically release all the children (Label)

    // don't forget to call "super dealloc"
    [super dealloc];
}
@end
```

Статическая функция `scene` создает объект типа `CCScene`, помещает в него наш класс `HelloWorld` (наследуемый от `CCLayer`) и возвращает созданный объект.

Функция `init` размещает в объекте нужные компоненты. В этом примере создается статическая метка `CCLabel`, которая добавляется в слой вызовом функции `addChild`. Таким образом, библиотека Cocos2D берет на себя всю "черную работу" по созданию компонентов, позволяя нам оперировать вполне читабельным кодом.

Теперь можно скомпилировать проект. Запускаем компиляцию… и получаем сообщение об ошибке. Дело в том, что мастер создал нужные классы, но исходные

тексты библиотеки мы должны добавить самостоятельно. В проекте есть папка libs, но она пуста. Для того чтобы скомпилировать проект, необходимо скопировать туда файлы из библиотеки, как показано в прилагаемом проекте. Теперь можно выполнить компиляцию еще раз (рис. 8.13).

Рис. 8.13. Проект HelloWorld, созданный с помощью библиотеки Cocos2D

Иерархическое представление интерфейса в Cocos2D

Вряд ли можно найти игровую программу, состоящую из одного окна. Поэтому нужно научиться переключать диалоговые окна. Как и в случае со стандартным интерфейсом iOS, окна будут полноэкранными. Для примера, добавим в наше приложение HelloWorld крайне важное окно **About.**

Первым делом нужно добавить на экран меню. Добавим в функцию `init` строки в листинге 8.12.

Листинг 8.12. Добавление меню

```
CGSize size = [[CCDirector sharedDirector] winSize];

// Создать текстовую метку
CCLabel* label = [CCLabel labelWithString:@"Hello World"
                          fontName:@"Marker Felt" fontSize:64];
label.position = ccp(size.width/2, size.height-label.contentSize.height);
label.color = ccGRAY;
[self addChild: label];

// About
CCMenuItemFont *menu_about = [CCMenuItemFont itemFromString:@"About"
                             target:self selector:@selector(menuAbout:)];
menu_about.position =  ccp(0, 20);

CCMenu *menu = [CCMenu menuWithItems: menu_about, nil];
[self addChild: menu];
```

Мы создали пункт меню, класс `CCMenu` и добавили меню на экран, передвинув надпись "Hello World" повыше, чтобы она не заслоняла меню, как на рис. 8.14. Наше меню состоит из единственного пункта **About**, но для понимания сути этого вполне достаточно.

Осталось добавить функцию-обработчик в листинге 8.13.

Рис. 8.14. Меню на главном окне программы

Листинг 8.13. Обработчик нажатия меню

```
- (void) menuAbout:(id) sender
{
    [[CCDirector sharedDirector] pushScene:[MainMenuAbout scene]];
}
```

Мы помещаем новую сцену с помощью вызова функции `CCDirector:: pushScene`. Теперь создадим класс дочернего окна **About**. Очевидно, что по своей структуре класс дочернего окна практически не будет отличаться от окна главного. Рассмотрим код в листинге 8.14.

Листинг 8.14. Код окна About

Файл HelloWorldScene.h:

```
@interface MainMenuAbout : CCLayer
{

}

+ (id) scene;
- (id) init;
- (void) menuBack:(id) sender;

@end
```

Файл HelloWorldScene.m:

```
@implementation MainMenuAbout

+ (id) scene
```

```
{
     // 'scene' is an autorelease object.
     CCScene *scene = [CCScene node];

     // 'layer' is an autorelease object.
     MainMenuAbout *layer = [MainMenuAbout node];

     // add layer as a child to scene
     [scene addChild: layer];

     // return the scene
     return scene;
}

- (id) init
{
     if( (self=[super init]))
     {
          CGSize s = [[CCDirector sharedDirector] winSize];

          // Фон
          NSString *s_img = @"ImgBack.png";
          CCSprite *back = [CCSprite spriteWithFile:s_img];
          back.position  = ccp(s.width/2, s.height/2);
          [self addChild:back z:-1];

          CCLabel* label = [CCLabel labelWithString:@"About:"
                              fontName:@"Marker Felt" fontSize:32];
          label.position =  ccp(50, s.height - 17);
          [self addChild: label];

          NSString *s_txt = @"Cocos 2D HelloWorld. (c) 2011 v1.0."
                            @" Created by Dmitry.";
          CCLabel* text = [CCLabel labelWithString:s_txt
                               dimensions:CGSizeMake(450, 120)
                               alignment:UITextAlignmentCenter
                               fontName:@"Marker Felt" fontSize:18];
          text.position =  ccp(10 + text.contentSize.width/2,
                              s.height - 140);
          [self addChild: text];

          // Кнопка Back
          CCMenuItemFont *game_back = [CCMenuItemFont itemFromString:
                                          @"Back" target:self
                                          selector:@selector(menuBack:)];
```

```
        CCMenu *menu = [CCMenu menuWithItems: game_back, nil];
        game_back.position   = ccp(s.width/2 - 35, s.height/2 - 17);
        [self addChild: menu];
    }

    return self;
}

- (void) menuBack: (id) sender
{
     [[CCDirector sharedDirector] popScene];
}

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

@end
```

В функции `init` мы создаем несколько объектов:

- фоновый рисунок, этот объект не является обязательным, но он делает окно более привлекательным;
- текст содержимого окна, отображаемый с помощью элемента `CCLabel`;
- кнопку **Back** с помощью элемента `CCMenuItemFont`.

В главном окне мы помещаем новую сцену с помощью функции `pushScene`, а когда она больше не нужна, убираем, вызвав `popScene` в функции `menuBack`.

Теперь можно скомпилировать программу и проверить переключение окон кнопкой **About**, как показано на рис. 8.15.

Соответственно при нажатии кнопки **Back** окно должно закрываться, а вместо него — снова появляться основной экран программы.

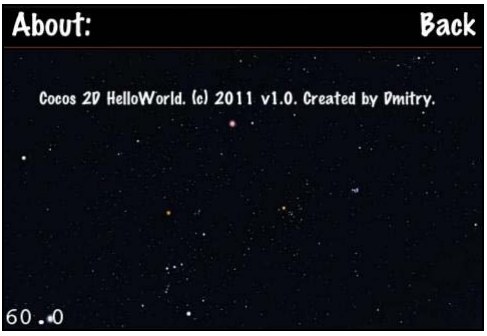


Рис. 8.15. Окно **About**

Элементы графического интерфейса Cocos2D

С некоторыми элементами интерфейса мы уже знакомы: это текстовые метки `CCLabel`, элементы меню `CCMenu` и изображения `CCSprite`. Рассмотрим их более подробно.

Допустим, мы хотим активировать нужный пункт меню или управлять положением спрайта на экране. Для этого нужно сохранить указатель на объект для последующего использования. Например, для элементов меню мы должны написать следующий код:

```
pMenuItemNew = [[CCMenuItemFont itemFromString:@"New game"
                                target:self
                                selector:@selector(menuGame01:)] retain];
pMenuItemContinue = [[CCMenuItemFont itemFromString:@"Continue game"
                                target:self
                                selector:@selector(menuGame02:)] retain];
```

(Соответствующие объекты `pMenuItemNew` и `pMenuItemContinue` должны быть объявлены в h-файле.)

Ключевое слово `retain` нам уже известно, оно увеличивает счетчик ссылок на единицу, позволяя сохранить объект для дальнейшего использования. Теперь мы можем, к примеру, деактивировать пункт меню с помощью свойства `isEnabled`:

```
pMenuItemNew.isEnabled = FALSE;
```

Мы можем убрать с экрана нужный пункт меню, если хотим показывать кнопку перехода на следующий уровень только по окончании игры:

```
pMenuItemContinue.visible = FALSE;
```

Чтобы показать меню, параметр `visible` нужно снова установить в `TRUE`. В конце работы класса мы должны освободить объект с помощью функции `release`, вызываемой в функции `dealloc`.

```
[pMenuItemNew release];
[pMenuItemContinue release];
```

Перемещение объектов

Рассмотрим аналогичный пример — движение объекта по экрану касанием. Как и в случае с меню, необходимо сохранить указатель на объект, который мы будем перемещать.

Для того чтобы добавить в программу возможность обработки касаний, нужно реализовать три функции: `ccTouchesBegan`, `ccTouchesMoved` и `ccTouchesEnded`. Также в функции `init` необходимо установить свойство `.isTouchEnabled`:

```
self.isTouchEnabled = TRUE;
```

После этого функции `ccTouches` будут вызываться корректно.

Анимация

Анимацию объекта можно осуществить двумя способами. Самый простой способ — это воспользоваться встроенными возможностями библиотеки Cocos2D.

Листинг 8.15. Движение объекта

```
CCSprite *pSprite = [CCSprite spriteWithFile:@"Image.png"];
[self addChild:pSprite z:-2];

int period = 10 + CCRANDOM_0_1()*500/100;
id action = [CCRotateBy actionWithDuration:period angle:360];
id seq = [CCSequence actions:action, nil];
[pSprite runAction: [CCRepeatForever actionWithAction:seq]];
```

Создаем в листинге 8.15 объект класса `CCRotateBy` для вращения объекта и запускаем анимацию как последовательность действий `CCSequence`. Помимо `CCRotateBy`, в файле CCIntervalAction.h описаны и другие классы — `CCMoveBy`, `CCJumpBy`, `CCBezierBy`, `CCScaleBy`, `CCFadeIn`, `CCFadeOut`. Комбинируя методы и объединяя их в последовательность с помощью класса `CCSequence`, можно выполнять движение объекта по траектории.

Вторым способом является самостоятельное использование таймера. Таймер уже встроен в библиотеку Cocos2D, так что достаточно лишь его активировать. Рассмотрим листинг 8.16.

Листинг 8.16. Использование таймера

```
// активация таймера (в нужном месте кода)
[self schedule: @selector(tick:)];

// Функция таймера
- (void) tick: (ccTime) dt
{
     // Расчет параметров угла и положения объекта
     float x, y, angle;
     …

     // Обновление положения на экране
     pSprite.position = ccp(x, y);
     pSprite.rotation = angle;
}
```

Встроенный макрос `ccp` позволяет задавать положение по двум координатам. Реализация алгоритма зависит от идеи (например, расчет полета ядра под действием силы тяжести и сопротивления воздуха).

Использование библиотеки Box2D

Прилагаемый с дистрибутивом Cocos2D визард позволяет создать готовый шаблон проекта XCode с применением обеих библиотек.

Код, необходимый для активации Box2D с помощью таймера и обновления объектов, показан в листинге 8.17.

Листинг 8.17. Использование таймера для работы библиотеки Box2D

```
- (void) tick: (ccTime) dt
{
     int32 velocityIterations = 8;
     int32 positionIterations = 1;
     pWorld->Step(dt, velocityIterations, positionIterations);

     // Обновить положение объекта
     float rx = pBody->GetPosition().x, ry = pBody->GetPosition().y;
     CGPoint obj_pos = ccp(10*rx, 10*ry);
     pSprite.position = obj_pos;
     pSprite.rotation = -CC_RADIANS_TO_DEGREES(pBody->GetAngle());
}
```

Мы сначала запускаем функцию `Step` для "активации" созданного мира, затем обновляем положение экранных объектов в соответствии с новыми данными. Для указания положения достаточно двух параметров — координат и угла направления.

Воспроизведение звука

Библиотека Box2D имеет встроенные средства воспроизведения звука, для этого используются классы `CDSoundSource`, `CDAudioManager` и `SimpleAudioEngine`. В описании последнего можно найти много полезных функций, например `playBackgroundMusic`, `playEffect` и пр., которые сделают игру более яркой и привлекательной. Рассмотрим пример, приведенный в листинге 8.18.

Листинг 8.18. Воспроизведение звука

```
CDSoundSource *pSound;

- (id) init
{
     …
     // Инициализация
     pSound = [[soundEngine soundSourceForFile:@"Engine.wav"] retain];
}

- (void) play
{
     // Воспроизвдение
     if (![pSound isPlaying])
          [pSound play];
}

- (void)dealloc
```

```
{
    …
    // Удаление
    [pSound release];

    [super dealloc];
}
```

Игровой проект

За основу возьмем пример на рис. 8.4. Переделаем его с помощью библиотеки Cocos2D, добавив дополнительные элементы, присущие "настоящей" игре.

Хватит трех экранов, а основное меню будет содержать только два пункта — **New game** и **About**:

- окно **About** будет выводить информацию об игре;
- окно выбора уровня будет открываться из главного меню;
- окно игры представляет игровой процесс.

Итак, приступим.

Главное меню

Главное меню должно содержать два пункта, но с ними окно будет казаться пустым. Поэтому добавим фоновый рисунок, чтобы несколько "оживить" интерфейс.

> ***Примечание***
>
> В реальных проектах изображения должны разрабатываться дизайнером: как показывает опыт, умение хорошо рисовать и программировать одновременно редко встречается.

Рис. 8.16. Главное меню программы

Здесь мы добавим еще звуковое сопровождение нажатия на кнопки меню. Нужно создать наследуемый класс и переопределить в нем функцию `selected`, как показано в листинге 8.19.

Листинг 8.19. Добавление звука в меню

Файл MenuItems.h:

```
@interface CCMenuItemFontWithSound : CCMenuItemFont
{
}
@end
```

Файл MenuItems.m:

```
@implementation CCMenuItemFontWithSound

- (void) selected
{
        [[GameSoundManager sharedManager].soundEngine
                                              playEffect:@"Click.wav"];

        [super selected];
}

@end
```

Теперь мы можем использовать этот класс меню вместо стандартного, и при нажатии пункта меню будет слышен звук.

Класс главного меню показан в листинге 8.20.

Листинг 8.20. Окно главного меню

Файл MainMenu.h:

```
@interface MainMenu : CCLayer
{
}

+ (id) scene;
- (id) init;
- (void) menuNew:(id) sender;
- (void) menuAbout:(id) sender;
@end
```

Файл MainMenu.mm:

```
#import "MainMenu.h"
#import "LevelSelect.h"
```

```
#import "About.h"

@implementation MainMenu

+ (id) scene
{
    // 'scene' is an autorelease object.
    CCScene *scene = [CCScene node];

    // 'layer' is an autorelease object.
    MainMenu *layer = [MainMenu node];

    // add layer as a child to scene
    [scene addChild: layer];

    // return the scene
    return scene;
}

- (id) init
{
    if( (self=[super init]))
    {
        CGSize s = [[CCDirector sharedDirector] winSize];

        // Фон
        NSString *s_img = @"ImgBack.png";
        CCSprite *back = [CCSprite spriteWithFile:s_img];
        back.position = ccp(s.width/2, s.height/2);
        [self addChild:back z:-1];

        NSString *s_txt = @"Cocos 2D Game. (c) 2011 v1.0";
        CCLabel* text = [CCLabel labelWithString:s_txt
                             dimensions:CGSizeMake(450, 120)
                             alignment:UITextAlignmentCenter
                             fontName:@"Marker Felt" fontSize:18];
        text.position =  ccp(160, 0);
        [self addChild: text];

        // Кнопка About
        CCMenuItemFontWithSound *menu_about = [CCMenuItemFontWithSound
                                itemFromString:@"About" target:self
                                selector:@selector(menuAbout:)];
        menu_about.position =  ccp(0, 20);
        // Кнопка New
```

```
        CCMenuItemFontWithSound *menu_new = [CCMenuItemFontWithSound
                                  itemFromString:@"New" target:self
                                  selector:@selector(menuNew:)];
        menu_new.position =  ccp(0, 60);

        CCMenu *menu = [CCMenu menuWithItems: menu_about, menu_new, nil];
        [self addChild: menu];
    }

    return self;
}

- (void) menuNew: (id) sender
{
     [[CCDirector sharedDirector] pushScene: [LevelSelect scene]];
}

- (void) menuAbout: (id) sender
{
     [[CCDirector sharedDirector] pushScene: [MainMenuAbout scene]];
}

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

@end
```

Окно **About** еще проще, его код будет содержать обработчик всего одной функции Back, как показано в листинге 8.21.

Листинг 8.21. Окно About

Файл About.h:

```
@interface MainMenuAbout : CCLayer
{
}

// returns a Scene that contains the HelloWorld as the only child
+ (id) scene;
- (id) init;
- (void) menuBack: (id) sender;

@end
```

Файл About.mm:

```
@implementation MainMenuAbout

+ (id) scene
{
    // 'scene' is an autorelease object.
    CCScene *scene = [CCScene node];

    // 'layer' is an autorelease object.
    MainMenuAbout *layer = [MainMenuAbout node];

    // add layer as a child to scene
    [scene addChild: layer];

    // return the scene d
    return scene;
}

- (id) init
{
    if( (self=[super init]))
    {
        CGSize s = [[CCDirector sharedDirector] winSize];

        // Фон
        NSString *s_img = @"ImgBack.png";
        CCSprite *back = [CCSprite spriteWithFile:s_img];
        back.position = ccp(s.width/2, s.height/2);
        [self addChild:back z:-1];

        CCLabel* label = [CCLabel labelWithString:@"About:"
                              fontName:@"Marker Felt" fontSize:32];
        label.position =  ccp(50, s.height - 20);
        [self addChild: label];

        // Информация о программе
        NSString *s_txt = @"Cocos 2D Game. (c) 2011 v1.0."
                          @"Created by Dmitry.";
        CCLabel* text = [CCLabel labelWithString:s_txt
                              dimensions:CGSizeMake(220, 300)
                              alignment:UITextAlignmentCenter
                              fontName:@"Marker Felt" fontSize:18];
        text.position =  ccp(160, 100);
        [self addChild: text];
```

```
        // Кнопка Back
        CCMenuItemFontWithSound *game_back = [CCMenuItemFontWithSound
                                itemFromString:@"Back" target:self
                                selector:@selector(menuBack:)];
        CCMenu *menu = [CCMenu menuWithItems: game_back, nil];
        game_back.position   = ccp(s.width/2 - 35, s.height/2 - 20);
        [self addChild: menu];
    }

    return self;
}

- (void) menuBack: (id) sender
{
     [[CCDirector sharedDirector] popScene];
}

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

@end
```

Окно содержит фоновый рисунок, текст, хранящийся в отдельной переменной (чтобы при необходимости его было легче заменить), и кнопку **Back**.

Следующим шагом необходимо сделать окно выбора уровня. В нашей игре уровень будет только один, остальные можно будет добавить позже при необходимости. Рассмотрим код в листинге 8.22.

Листинг 8.22. Окно выбора уровня

Файл LevelSelect.h:

```
@interface LevelSelect : CCLayer
{
       // Количество уровней
       #define COUNT 5
       CCMenuItemFontWithSound *levels[COUNT];
}

+ (id) scene;
- (id) init;

@end
```

Файл LevelSelect.mm:

```
#import "LevelSelect.h"
#import "LevelGame.h"

int nLevelAvailable = 1; // уровень, доступный для игрока

@implementation LevelSelect

+ (id) scene
{
    CCScene *scene = [CCScene node];
    LevelSelect *layer = [LevelSelect node];
    [scene addChild: layer];
    return scene;
}

- (id) init
{
    if( (self=[super init]))
    {
        CGSize s = [[CCDirector sharedDirector] winSize];

        // Фон
        NSString *s_img = @"ImgBack1.png";
        CCSprite *back = [CCSprite spriteWithFile:s_img];
        back.position = ccp(s.width/2, s.height/2);
        [self addChild:back z:-1];

        CCLabel* label = [CCLabel labelWithString:@"Select level:"
                              fontName:@"Marker Felt" fontSize:32];
        label.position =  ccp(80, s.height - 20);
        [self addChild: label];

        // Пункты меню
        CCMenuItemFontWithSound *game_back = [CCMenuItemFont
                                itemFromString:@"Back" target: self
                                selector:@selector(menuBack:)];
        levels[0] = [[CCMenuItemFontWithSound itemFromString: @"Level 01"
                   target:self selector:@selector(menuLevel01:)] retain];
        levels[1] = [[CCMenuItemFontWithSound itemFromString: @"Level 02"
                   target:self selector:@selector(menuLevel02:)] retain];
        levels[2] = [[CCMenuItemFontWithSound itemFromString: @"Level 03"
                   target:self selector:@selector(menuLevel03:)] retain];
        levels[3] = [[CCMenuItemFontWithSound itemFromString: @"Level 04"
                   target:self selector:@selector(menuLevel04:)] retain];
```

```
        levels[4] = [[CCMenuItemFontWithSound itemFromString: @"Level 05"
                    target:self selector:@selector(menuLevel05:)] retain];

        for(int p=0; p<COUNT; p++)
        {
            levels[p].isEnabled = p == 0;
            levels[p].position = ccp(0, 150 - 40*p);
        }

        CCMenu *menu = [CCMenu menuWithItems: game_back, levels[0],
                     levels[1], levels[2], levels[3], levels[4], nil];
        game_back.position   = ccp(s.width/2 - 35, s.height/2 - 20);
        [self addChild: menu];
    }

    return self;
}

- (void) menuBack: (id) sender
{
     [[CCDirector sharedDirector] popScene];
}

- (void) menuLevel01: (id) sender
{
     [[CCDirector sharedDirector] pushScene: [LevelGame scene:1]];
}

- (void) menuLevel02: (id) sender
{
     [[CCDirector sharedDirector] pushScene: [LevelGame scene:2]];
}

- (void) menuLevel03: (id) sender
{
     [[CCDirector sharedDirector] pushScene: [LevelGame scene:3]];
}

- (void) menuLevel04: (id) sender
{
     [[CCDirector sharedDirector] pushScene: [LevelGame scene:4]];
}

- (void) menuLevel05: (id) sender
{
```

```
     [[CCDirector sharedDirector] pushScene: [LevelGame scene:5]];
}

-(void) onEnter
{
    [super onEnter];
    // Обновление пунктов меню
    for(int p=0; p<COUNT; p++)
    {
        levels[p].isEnabled = (p+1 <= nLevelAvailable);
    }
}

- (void)dealloc
{
    for(int p=0; p<COUNT; p++)
         [levels[p] release];

    [super dealloc];
}

@end
```

Создаваемые пункты меню сохраняются в массиве с увеличением счетчика ссылок посредством функции `retain`. Это нужно для манипуляций различными пунктами меню по мере прохождения игры.

Рис. 8.17. Меню выбора уровня

Для обновления списка доступных уровней используется функция `onEnter`. Она вызывается во время активации окна, например после завершения игры и возврата в окно выбора уровня. В начале игры доступен только первый пункт, но затем, по мере прохождения уровней, пользователю могут быть доступны следующие уровни. В нашем примере номер пройденного уровня хранится в глобальной переменной. В реальном проекте следовало бы сохранять этот номер в настройках программы, чтобы он не сбрасывался при закрытии приложения.

Функция `scene` класса `LevelGame` отличается от ранее описанных аналогичных функций: при инициализации класса в него дополнительно передается номер уровня. Окно выбора уровня показано на рис. 8.17.

Рассмотрим основной класс игры в листинге 8.23 (не будем приводить его здесь целиком, лишь основные моменты).

Листинг 8.23. Основное окно игры

Класс LevelGame.h:

```
class MyContactListener; // этот класс рассматривался ранее

@interface LevelGame : CCLayer
{
        int nLevel;

        // Поворот объекта
        BOOL bPressed;
        CGFloatmfGunAngle, mfStartAngle;

        // Объекты
        CCSprite        *pSpriteGun, *pSpriteTarget;
        // Всплывающее окно
        CCSprite        *pPopupWnd;
        CCLabel       *pLabelInfo;
        CCMenuItemFontWithSound *pMenuRestart;
        // Звук
        CDSoundSource *pSoundFire;

        // Типы элементов
        #define BODY_GROUND 1
        #define BODY_BULLET 2
        #define BODY_BULLET_BLOW 20
        #define BODY_TARGET 3
        #define BODY_GUN     4

        // Box2D
        b2World *pWorld;
        b2Body  *pBodyTarget;
        MyContactListener *pContactListener;
```

```
        #define MAX_COUNT 100
        b2Body        *pBodyBullets[MAX_COUNT];
        CCSprite      *pSpriteBullets[MAX_COUNT];
}

@property int nLevel;

+ (id) scene: (int)level;
- (id) init;
- (void) initData;
- (void) menuBack: (id) sender;
- (void) menuRestart: (id) sender;
- (void) menuFire: (id) sender;

- (void) tick: (ccTime) dt;
- (void)ccTouchesBegan:(NSSet*)touches withEvent:(UIEvent*)event;
- (void)ccTouchesMoved:(NSSet*)touches withEvent:(UIEvent *)event;
- (void)ccTouchesEnded:(NSSet*)touches withEvent:(UIEvent *)event;

- (void) showPopupWnd;
- (void) hidePopupWnd;

- (CGFloat) AngleFromPt: (CGPoint)pt: (CGPoint)center;

@end
```

Класс LevelGame.mm:

```
#import "LevelGame.h"

#define PTM_RATIO 10

extern int nLevelAvailable;

@implementation LevelGame
@synthesize nLevel;

+ (id) scene: (int)level;
{
    CCScene *scene = [CCScene node];

    LevelGame *layer = [LevelGame node];
    layer.nLevel = level;
    [layer initData]; // инициализация данных уровня

    [scene addChild: layer];
    return scene;
}
```

```
- (id) init
{
    if( (self=[super init]))
    {
    }

    return self;
}

- (void) initData
{
    CGSize s = [[CCDirector sharedDirector] winSize];

    // Фон
    NSString *s_img = @"ImgBack1.png";
    CCSprite *back = [CCSprite spriteWithFile:s_img];
    back.position    = ccp(s.width/2, s.height/2);
    [self addChild:back z:-1];

    mfGunAngle = 0;

    // Спрайт пушки
    int x_pos = 0, y_pos = 280;
    pSpriteGun = [[CCSprite spriteWithFile:@"ImgGun.png"] retain];
    pSpriteGun.position     = ccp(x_pos + pSpriteGun.contentSize.width/2,
                    s.height - y_pos - pSpriteGun.contentSize.height/2);
    [self addChild:pSpriteGun];

    // Мишень
    int t_x = 50, t_y = 360;
    pSpriteTarget = [[CCSprite spriteWithFile:@"ImgTarget.png"] retain];
    pSpriteTarget.position = ccp(t_x+pSpriteTarget.contentSize.width/2,
                   s.height-t_y- pSpriteTarget.contentSize.height/2);
    [self addChild:pSpriteTarget];

    // Окно завершения игры
    pPopupWnd = [[CCSprite spriteWithFile:@"WndPopupBack.png"] retain];
    pPopupWnd.position      = ccp(s.width/2, s.height/2);
    [self addChild:pPopupWnd z:1];
    // Game Finish wnd - label
    pLabelInfo = [[CCLabel labelWithString:@"Level complete"
                    fontName:@"Marker Felt" fontSize:32] retain];
    pLabelInfo.position =  ccp(160, s.height - 190);
    pLabelInfo.color = ccGRAY;
    [self addChild: pLabelInfo z:1];
    // Game Finish wnd - restart game
```

```
    pMenuRestart = [[CCMenuItemFontWithSound itemFromString: @"Restart"
                  target: self selector:@selector(menuRestart:)] retain];
    pMenuRestart.position   = ccp(0, -40);


    // Звук выстрела
    pSoundFire = [[[GameSoundManager sharedManager].soundEngine
                             soundSourceForFile:@"Fire.wav"] retain];

    // Box2D
    b2Vec2 gravity(0.0f, 0.0f);
    bool doSleep = true;
    pWorld = new b2World(gravity, doSleep);

    // Проверка столкновений
    pContactListener = new MyContactListener();
    pWorld->SetContactListener(pContactListener);


    // Список снарядов
    for(int p=0; p<MAX_COUNT; p++)
    {
        pBodyBullets[p]   = NULL;
        pSpriteBullets[p] = nil;
    }

    pPopupWnd.visible = FALSE;
    pLabelInfo.visible = FALSE;
    pMenuRestart.visible = FALSE;

    // активизация касаний экрана
    self.isTouchEnabled = YES;

     [self schedule: @selector(tick:)];
}

- (void) tick: (ccTime) dt
{
    int32 velocityIterations = 8;
    int32 positionIterations = 1;
    pWorld->Step(dt, velocityIterations, positionIterations);

    pSpriteTarget.position = ccp(PTM_RATIO*pBodyTarget->GetPosition().x,
                                 PTM_RATIO*pBodyTarget->GetPosition().y);

    // Вывод и проверка столкнувшихся объектов
    for(int p=0; p<MAX_COUNT; p++)
```

```
    {
        if (pBodyBullets[p] != NULL && pSpriteBullets[p] != nil)
        {
            pSpriteBullets[p].position = ccp(PTM_RATIO*pBodyBullets[p]->
             GetPosition().x,PTM_RATIO*pBodyBullets[p]->GetPosition().y);
            pSpriteBullets[p].rotation = -
                    CC_RADIANS_TO_DEGREES(pBodyBullets[p]->GetAngle());
        }

        // Столкновения
        if (pBodyBullets[p] != nil &&
            (int)pBodyBullets[p]->GetUserData() == BODY_BULLET_BLOW)
        {
            // Удалить снаряд
            pWorld->DestroyBody(pBodyBullets[p]);
            pBodyBullets[p] = NULL;
            // Спрятать спрайт
            pSpriteBullets[p].visible = FALSE;

            // Сделать доступным следующий уровень
            if (self.nLevel + 1 >= nLevelAvailable)
                nLevelAvailable = self.nLevel + 1;
            // Конец игры
            [self showPopupWnd];
            break;
        }
    }
}

- (void) showPopupWnd
{
    pPopupWnd.visible = TRUE;
    pLabelInfo.visible = TRUE;
    pMenuRestart.visible = TRUE;
}

- (void) hidePopupWnd
{
    pPopupWnd.visible = FALSE;
    pLabelInfo.visible = FALSE;
    pMenuRestart.visible = FALSE;
}

- (void) menuFire: (id) sender
{
    [pSoundFire play];
```

```
    // Положение
    CGRect  rect = [pSpriteGun boundingBox];
    CGPoint center = CGPointMake(CGRectGetMidX(rect),
                                 CGRectGetMidY(rect));
    center = [[CCDirector sharedDirector] convertToGL:center];
    float angle = CC_DEGREES_TO_RADIANS(pSpriteGun.rotation);


    CGSize s = [[CCDirector sharedDirector] winSize];
    float x = center.x + 100*cos(angle);
    float y = s.height - center.y - 100*sin(angle);

    // Найти свободный элемент массива
    for(int p=0; p<MAX_COUNT; p++)
    {
        if (pBodyBullets[p] == nil)
        {
            // Спрайт
            if (pSpriteBullets[p] == nil)
            {
                pSpriteBullets[p] = [[CCSprite
                          spriteWithFile:@"ImgBullet.png"] retain];
                pSpriteBullets[p].visible = FALSE;
                [self addChild:pSpriteBullets[p]];
            }

            // Снаряд
            b2BodyDef bodyDef;
            bodyDef.type = b2_dynamicBody;
            bodyDef.bullet = TRUE;
            bodyDef.position.Set(x/PTM_RATIO, y/PTM_RATIO);
            bodyDef.userData = (void*)BODY_BULLET;
            pBodyBullets[p] = pWorld->CreateBody(&bodyDef);

            // Показать
            pSpriteBullets[p].visible = TRUE;
            break;
        }
    }
}
- (void)ccTouchesMoved:(NSSet *)touches withEvent:(UIEvent *)event
{
    if (bPressed)
    {
        for( UITouch *touch in touches )
        {
            CGPoint pt = [touch locationInView: [touch view]];
```

```
            pt = [[CCDirector sharedDirector] convertToGL: pt];

            CGRect rect = [pSpriteGun boundingBox];
            CGPoint center = CGPointMake(CGRectGetMidX(rect),
                                         CGRectGetMidY(rect));

            mfGunAngle = [self AngleFromPt: pt: center] - mfStartAngle;
            pSpriteGun.rotation = -CC_RADIANS_TO_DEGREES(mfGunAngle);
            break;
        }
    }
}

@end
```

Для вращения орудия служит обработка касаний экрана. Для хранения "снарядов" используются два массива объектов — массив спрайтов и массив "тел". Синхронизацию спрайтов и физических данных необходимо делать в функции таймера.

Вывод всплывающего окна окончания игры обеспечивается с помощью трех дополнительных объектов: картинки с фоновым изображением окна, текста и пункта меню (напомним, что встроенная поддержка каких-либо окон отсутствует в Cocos2D).

Переменная номера уровня инициализируется, но нигде не используется. Пользователи самостоятельно могут добавить поддержку данных для разных уровней, для чего следует изменить код функции `initData` (загружать различные фоновые рисунки для разных уровней, создавать различное количество целей и пр.).

Теперь можно скомпилировать программу и запустить фактически готовую игру (рис. 8.18).

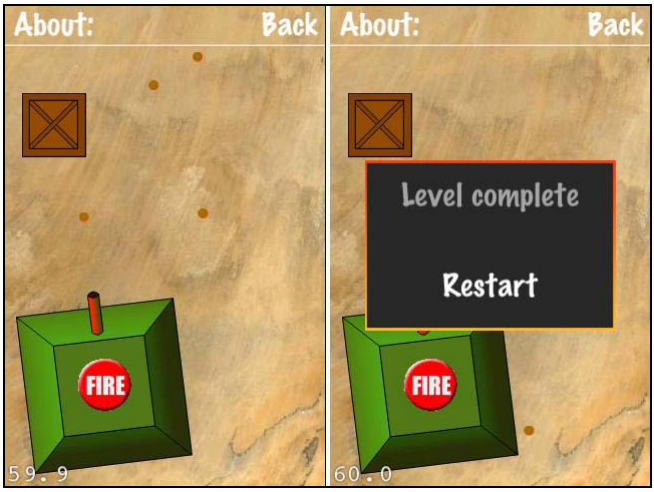


Рис. 8.18. Основное игровое окно

Пушка появляется на экране в виде спрайта, но для нее не создано объекта в Box2D. Она "невидима" для библиотеки, и, отражаясь от стен, снаряды свободно проходят сквозь корпус пушки. Есть два варианта решения проблемы: либо взрывать снаряды при ударе о стену, либо добавить новый объект для пушки в Box2D.

Поддержка поворота экрана

Готовое приложение должно поддерживать поворот экрана, и придется добавить его с помощью класса `NSNotificationCenter`. Необходимый код разместим в функции `applicationDidFinishLaunching`, как показано в листинге 8.24.

Листинг 8.24. Поддержка разворота экрана в классе `Cocos2DGameAppDelegate`

```
- (void) applicationDidFinishLaunching:(UIApplication*)application
{
     // CC_DIRECTOR_INIT()
     …
      [[UIDevice currentDevice]
                beginGeneratingDeviceOrientationNotifications];
      [[NSNotificationCenter defaultCenter] addObserver:self
                    selector:@selector(orientationChanged:)
                    name:@"UIDeviceOrientationDidChangeNotification"
                    object:nil];

     // Активацию основного окна делаем в функции orientationChanged
     bFirstRun = TRUE;
}

-(void) orientationChanged:(NSNotification *)notification
{
     UIDeviceOrientation orientation = [[UIDevice currentDevice]
                                       orientation];

     if (orientation == UIDeviceOrientationLandscapeLeft ||
         orientation == UIDeviceOrientationPortrait)
        [[CCDirector sharedDirector]
            setDeviceOrientation:kCCDeviceOrientationPortrait];
     if (orientation == UIDeviceOrientationLandscapeRight ||
         orientation == UIDeviceOrientationPortraitUpsideDown)
        [[CCDirector sharedDirector]
            setDeviceOrientation:kCCDeviceOrientationPortraitUpsideDown];

     if (bFirstRun)
     {
```

```
            bFirstRun = FALSE;
            [[CCDirector sharedDirector] runWithScene: [MainMenu scene]];
        }
}
```

Функция `addObserver` подписки на сообщения поворота экрана сохраняет с помощью функции `orientationChanged` новое значение поворота в Cocos2D, вызывая `CCDirector: setDeviceOrientation`. Класс окна создан лишь после того, как ориентация устройства установлена. Переменная `bFirstRun` используется для того, чтобы функция `CCDirector::runWithScene` вызвалась лишь один раз и не активировалась в дальнейшем при поворотах устройства.

На этом мы заканчиваем рассмотрение технологий создания игровых программ и отсылаем читателя к странице **http://code.google.com/p/cocos2d-iphone/wiki/GamesUsingCocos2d**, где приведены разнообразные примеры.

А мы перейдем к не менее интересной части — продаже программных продуктов в Apple App Store.

Глава 9

Продажа программ в iOS — основные принципы

Предложить программу пользователям не менее важно, чем ее написать. Магазин Apple App Store автоматизирует распространение и продажу ПО, фактически избавляя разработчика от лишних хлопот.

Виды моделей распространения программного обеспечения в Apple iOS

Те, кто хоть раз пытался официально купить любую программу для настольного компьютера, помнят, насколько непростым может быть этот процесс. Для того чтобы защититься от пиратства, разработчики придумывали различные схемы, от пробного периода до онлайн-активации через сеть Интернет, не говоря уже об аппаратных ключах, подключаемых через USB-порт. Сложно сказать, реально ли это помогало, или скорее мешало законопослушным пользователям (как известно, чем сложнее защита, тем больше вероятность сбоев в ее работе). Так вот, для разработчиков iOS есть хорошая новость: ничего этого здесь не нужно. Компания Apple полностью берет на себя весь процесс доставки программы к клиенту, от оплаты продукта до инсталляции программного обеспечения.

Всего существует несколько вариантов распространения и оплаты программного обеспечения в iOS:

- ❒ бесплатные программы — самый простой вариант;
- ❒ платные приложения — разработчик указывает цену программы, все остальное делает Apple;
- ❒ in-app purchase — позволяет расширить функциональность программы, предлагая покупателю различные платные услуги:
  - разовая покупка (например, доступ к дополнительным уровням игры);
  - многократная покупка (например, покупка единиц оружия в многопользовательской игре);
  - подписка (например, доступ к новым номерам журнала в течение полугода).

  Использование in-app purchase требует написания дополнительного кода и выходит за рамки этой книги, желающие могут найти в сети Интернет статью "Интегрируем In-App Purchases", размещенную на сайте **http://habrahabr.ru**;

- интеграция в программу рекламных баннеров (iAd) — начиная с iOS 4.0, разработчик может предложить пользователям несколько версий, например платную без рекламы и бесплатную с баннерами.

А сейчас мы выясним, как осуществляется покупка программы конечным пользователем.

Инсталляция и деинсталляция программ в iOS

Прямо на "рабочем столе" устройства имеется значок App Store, открывающий доступ к каталогу программ. Принципиально важно, что пользователь iOS может устанавливать приложения только с помощью App Store (он не имеет доступа к файловой системе), поэтому разработчик уверен, что получает максимально возможную аудиторию. Для сравнения, на настольных компьютерах количество интернет-каталогов программ исчисляется сотнями, и разработчикам программ приходится даже платить за специальные сервисы, которые размещают программу на десятках различных каталогов. Впрочем, эффективность даже в этом случае остается низкой.

Внешний вид каталога на iPad показан на рис. 9.1 (на iPhone размер экрана меньше). Кроме категорий, по которым отсортированы программы, доступна строка поиска.

Рис. 9.1. Каталог программ App Store

Рис. 9.2. Поиск программы в App Store

Рис. 9.3. Программа в App Store

Попробуем найти в каталоге программу, отображающую фазы Луны. Набираем в строке поиска `moon phase` и получаем несколько программ, как показано на рис. 9.2.

Для каждой программы отображается значок и цена.

Примечание

Значок — это первое, что видит в процессе поиска покупатель, поэтому он должен выглядеть действительно привлекательно. Неряшливый значок может оттолкнуть часть потенциальных покупателей еще до момента перехода к странице программы.

Щелкаем понравившийся значок и видим страницу программы, как на рис. 9.3.

На странице мы видим три основных раздела:

- значок программы слева;
- описание программы;
- скриншоты программы.

Для регистрации собственной программы нужно предоставить всю эту информацию (об этом будет сказано в соответствующей главе).

Для инсталляции программы щелкнем кнопку **INSTALL APP**: на рабочем столе устройства появится новый значок, прямо под ним индикатор загрузки приложения. Все просто и элегантно. Не нужно вводить никаких ключей, все цифровые подписи автоматически копируются на устройство. Если программа была платной, то соответствующая сумма просто снимается с банковской карты, которую можно указать при регистрации учетной записи. Помимо банковской карты есть и другие способы оплаты приложения: подарочные карты либо промо-коды, которые разработчики могут выдавать в небольшом количестве бесплатно, например для тестирования программы.

Тем временем программа не заработала, выдав ошибку соединения с сервером. Видимо, фазы Луны в ней не рассчитывались (а просто загружались фотографии с какого-то сервера), и со спокойной совестью эту программу можно деинсталлировать продолжительным нажатием на значок программы (чтобы напротив нее появился значок удаления).

Примечание

В предыдущих версиях iOS в процессе удаления программы появлялось всплывающее окно, позволяющее указать рейтинг программы от 1 до 5. Однако у разработчиков было много нареканий на этот механизм: обычно деинсталлируют программу, которая не понравилась по каким-либо причинам, а полезную программу оставляют. Получалось так, что пользователи, которым программа не подошла, ставили низкие оценки, а остальные (которые успешно пользовались программой) не ставили никаких. В итоге рейтинг программы получал заметный перевес в сторону отрицательных значений. В последней версии iOS никакого окна уже не появляется. Впрочем, пользователи все равно могут оставить отзывы на странице приложения, для этого разработчикам рекомендуется вставлять ссылку на страницу отзывов прямо в приложение.

Стоит сказать о важном аспекте продажи программ для iOS — ценовой политике. Средняя стоимость программы для iOS составляет… 1−2 доллара. Более сложные программы (например, текстовый редактор Pages, созданный в Apple) стоят 9 долларов, но они, скорее, исключение. Если раньше типичная shareware-программа

стоила 19,99 доллара, и многим было проще поискать взломанную версию, то ради одного доллара смысла в этом уже нет, гораздо проще и быстрее поставить программу легальным способом.

Регистрация разработчика в App Store

Основным адресом для начала работы с сервисами Apple является сайт **http://developer.apple.com**. Для регистрации и получения сертификата разработчика нужно выполнить несколько шагов.

На сайте Apple выбрать программу iOS Developer Program, регистрация в которой стоит (на момент написания книги) 99 долларов в год.

Выбрать тип регистрации, как показано на рис. 9.4 — Individual или Company. В первом случае в разделе **seller** будет указано имя разработчика, во втором случае — название компании. Стоит ответственно подойти к выбору типа учетной записи, изменить его возможности не будет, в случае необходимости придется создавать новый.

Далее мы будем рассматривать регистрацию индивидуального разработчика.

1. Ввести личные данные, необходимые для оплаты членства с помощью банковской карты.

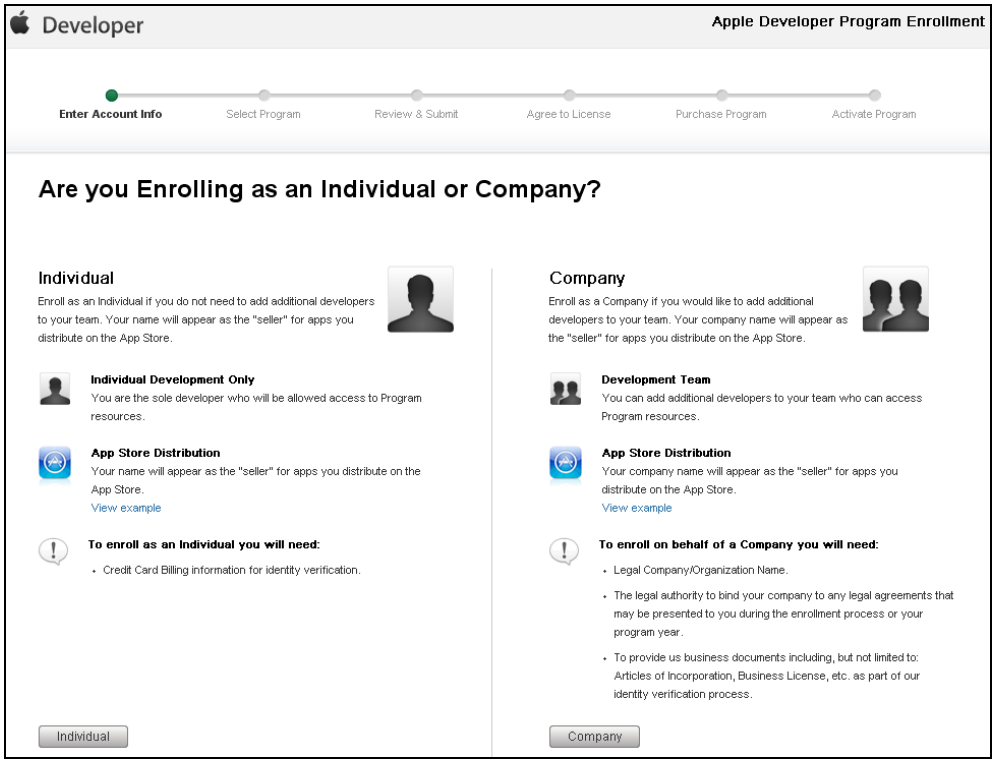


Рис. 9.4. Выбор типа регистрации

Важное замечание

Кредитная карта используется не только для оплаты, но и для подтверждения личности. Поэтому необходимо иметь собственную банковскую карту, на то же имя, что учетная запись. Регистрация счета обычно занимает 1–2 недели, так что процесс оформления карты можно начать, не дожидаясь активации учетной записи.

2. После ввода личных данных необходимо выбрать тип программы: iOS Developer Program (сюда входит iPhone и iPad), Mac Developer Program и Safari Developer Program. Нас интересует первый вариант. Далее следует согласиться с лицензионным соглашением.
3. Оплатить регистрацию и активировать учетную запись, что для российских разработчиков не так просто (рис. 9.5).

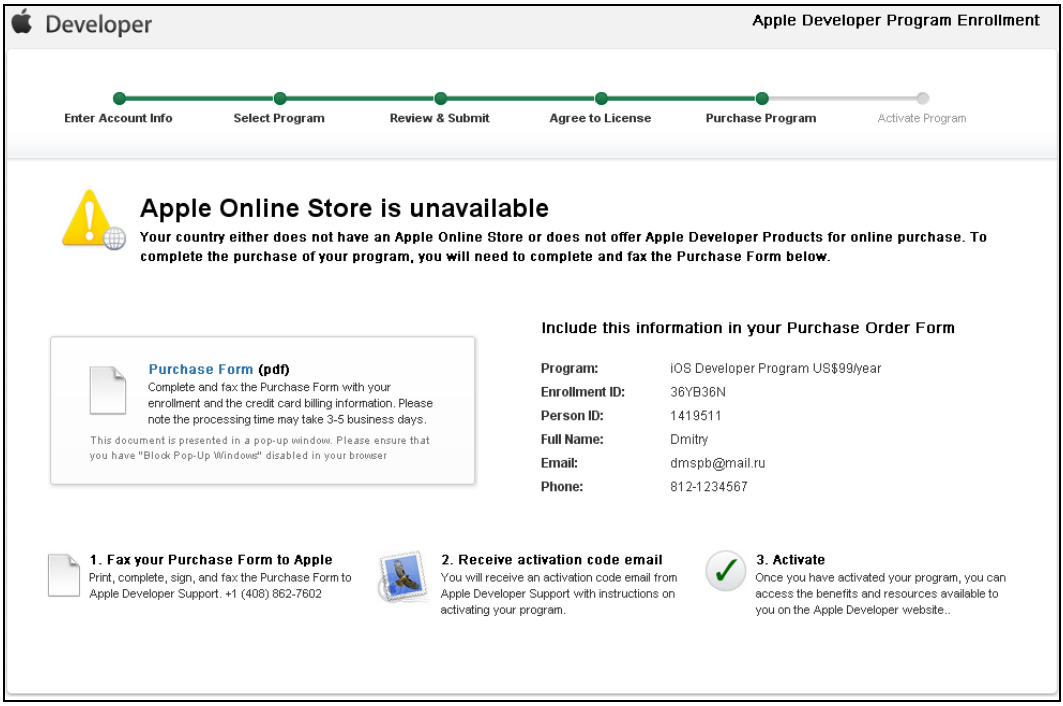


Рис. 9.5. Окно завершения регистрации

Как следует из текста, оплата через сайт Apple в России недоступна, поэтому нужно распечатать документ по прилагаемой ссылке **http://devimages.apple.com/programs/purchaseform.pdf**, вписать в него необходимые данные и отправить документ факсом в США на номер +1 (408) 862 7602, указанный на странице регистрации. Автор воспользовался услугами сайта **http://faxzero.com**, с помощью которого послать факс можно бесплатно. Для этого документ даже не пришлось печатать, было достаточно сохранить его на компьютере и вписать недостающие данные.

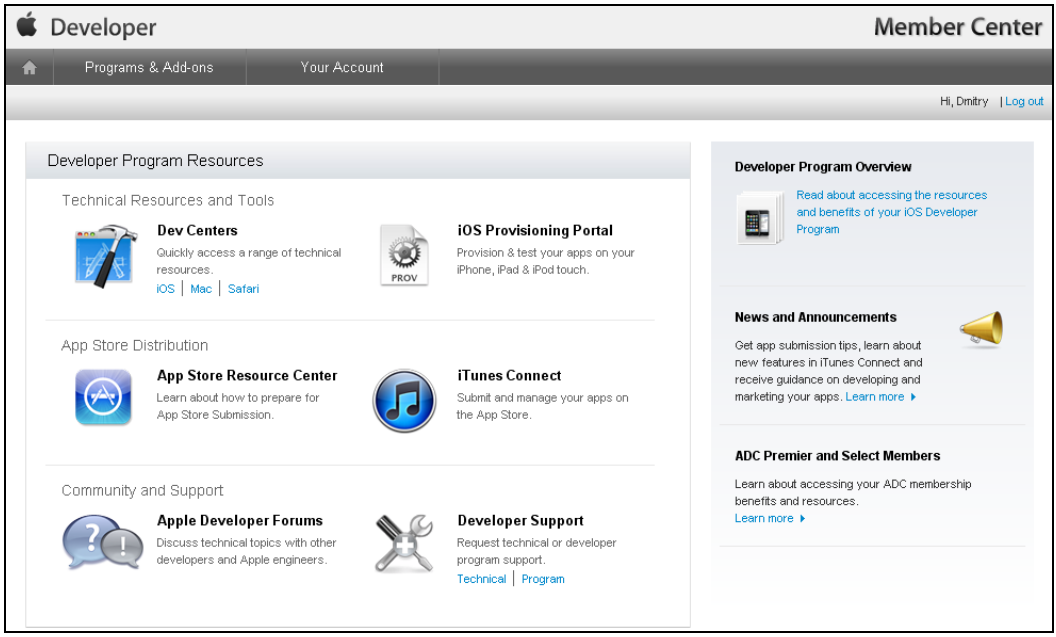


Рис. 9.6. Панель управления

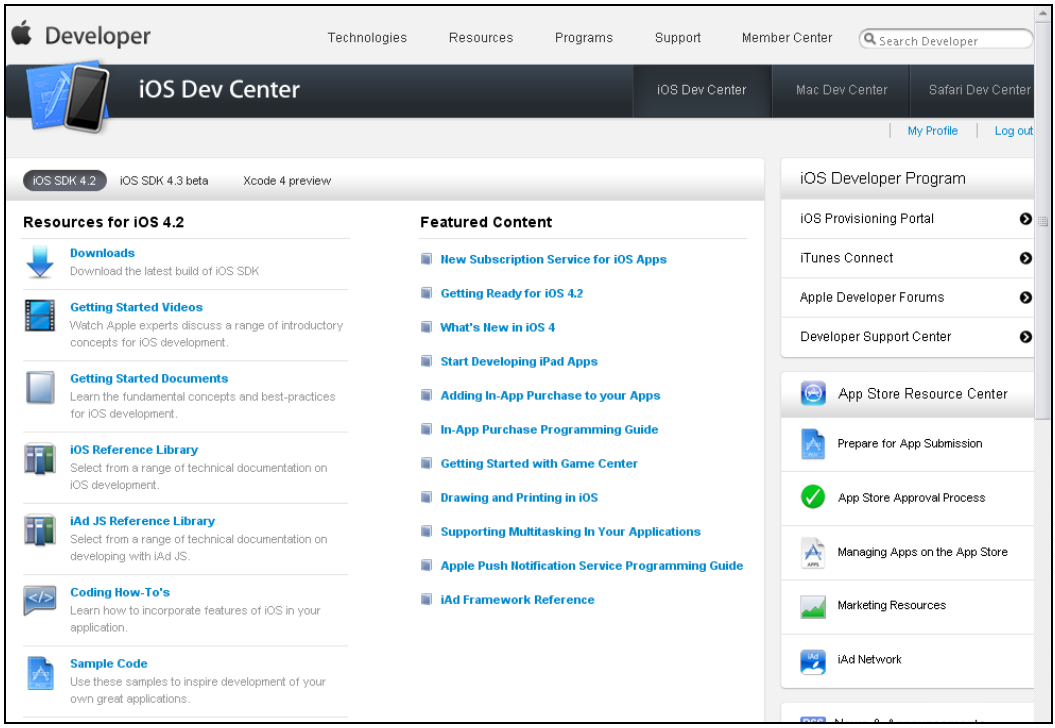


Рис. 9.7. Окно **iOS Dev Center**

На этом процедура регистрации завершена. Через несколько дней с банковской карты будут сняты деньги, а по почте придет подтверждение активации. Теперь разработчик может размещать программы в App Store.

Внешний вид панели управления показан на рис. 9.6.

Рассмотрим технические ресурсы и инструменты, которые предоставляет нам Apple.

Раздел *Dev Centers*

Как можно видеть из рис. 9.7, здесь собрана всевозможная документация: раздел **iOS Reference Library**, описывающий функции операционной системы, информация для начинающих в разделе **Getting Started Documents**, весьма полезный раздел с примерами **Sample Code** и т. д.

Раздел *iOS Provisioning Portal*

Этот раздел содержит данные, необходимые для тестирования и продажи программ в App Store (рис. 9.8). Здесь хранятся цифровые сертификаты, идентификаторы выпускаемых программ и всех устройств, на которых производится разработка. Любое добавление новой программы или нового устройства для разработки производится именно здесь.

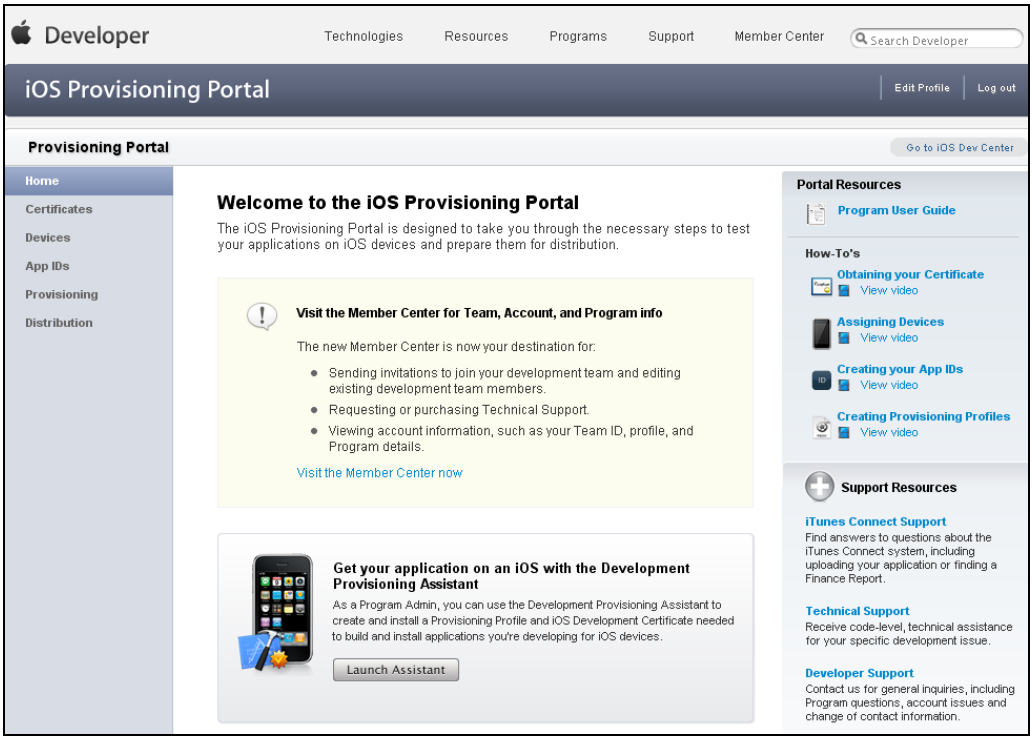


Рис. 9.8. Apple Provisioning Portal

Рассмотрим подразделы более подробно.

- **Certificates** — здесь хранятся сертификаты, которыми будут подписываться все программы, загружаемые на устройство или на сайт Apple. Для создания сертификата необходимо сгенерировать ключ (с помощью утилиты Связка ключей), затем загрузить его на сайт. Будет сгенерирован сертификат, который необходимо сохранить на компьютере, XCode будет автоматически подписывать им программы во время компиляции.

  Раздел **Certificates** состоит из двух частей — **Development** (Разработка) и **Distribution** (Распространение). Как можно догадаться из названия, первый сертификат используется для отладки программ и загрузки на устройство, второй сертификат используется для отправки программ в App Store или другим пользователям на тестирование;
- **Devices** — раздел содержит идентификаторы устройств, на которых пользователь может отлаживать и запускать программы. Идентификатор устройства представляет собой 40-значный код, узнать который можно с помощью iTunes (код устройства представляет собой шестнадцатеричное число, например bb33faa874f26c4527485094418249ffc6211d53);
- **App IDs** — раздел содержит идентификаторы всех выпускаемых программ. Уникальный идентификатор хранится в файле plist. Если мы откроем его для любой из созданных нами программ, то увидим примерно такие строки:

  ```
  <key>CFBundleIdentifier</key>
  <string>com.yourcompany.${PRODUCT_NAME:rfc1034identifier}</string>
  ```

  Этот идентификатор создан XCode и вполне подходит для отладки. Однако для выпуска коммерческих приложений необходимо создать уникальный идентификатор (рекомендуется использовать обратную нотацию, например `com.mysite.mysuperapp`). После того как программа выпущена, изменить ее идентификатор уже нельзя;
- **Provisioning** — раздел содержит цифровые сертификаты, используемые для приложений (рис. 9.9):
  - **Development** (Разработка) — действуют около 2 месяцев, потом их необходимо создавать заново;
  - **Distribution** (Распространение) — используется для подписания программы перед отправлением в App Store, неподписанную программу загрузить туда невозможно.

  Как для разработки, так и для распространения необходима пара сертификатов: общий сертификат из раздела **Certificates** и сертификат, принадлежащий конкретной программе, из раздела **Provisioning**. Перед компиляцией в настройках проекта необходимо выбрать сертификат **Development** или **Distribution** в разделе **Code signing** (Подписание), в зависимости от того что нужно, разработка или размещение в App Store;
- **Distribution** — содержит справочную информацию о распространении программ, никаких действий над программами здесь не выполняется.

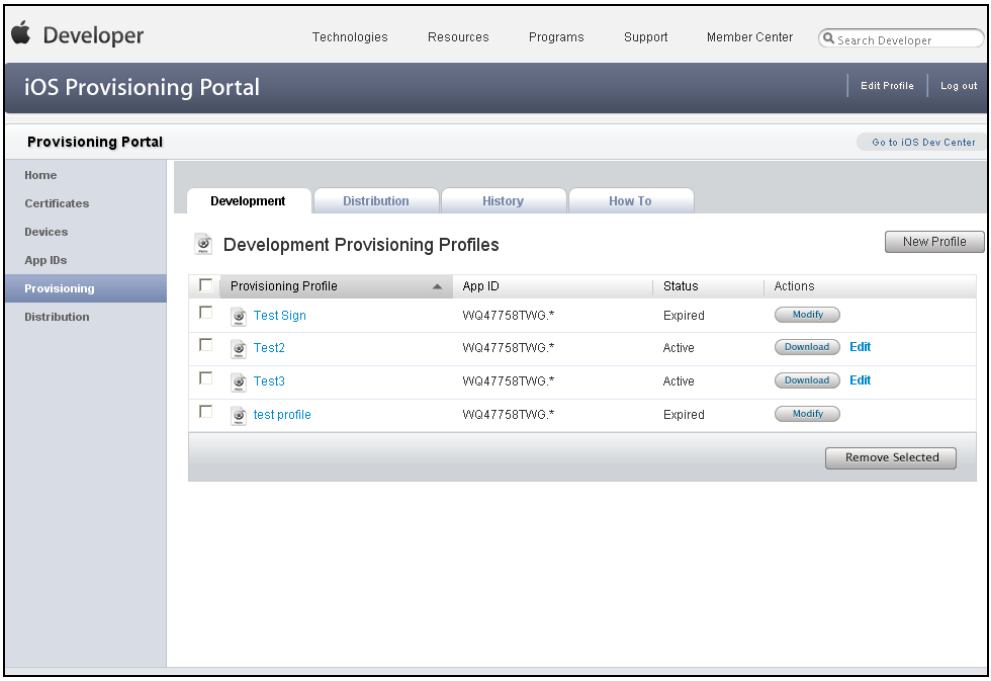


Рис. 9.9. Apple Provisioning Portal

Раздел *App Store Distribution*

Справочный раздел **App Store Resource Center** содержит документацию, необходимую для загрузки программ в App Store и управления загруженными программами.

Раздел **iTunes Connect** предлагает инструменты управления программами, загруженными в App Store (рис. 9.10):

- **Sales and Trends** — графическое представление продаж программ (рис. 9.11). Для владельцев iPhone и iPad рекомендуется скачать программу iTunes Connect Mobile, которая выводит эти же данные в более удобном виде;
- **Contracts, Tax, and Banking** — раздел необходимо заполнить в первую очередь. Он содержит "контракты" (лицензионное соглашение с Apple, с которым нужно согласиться) и банковскую информацию о названии банка и номере счета. Как раз сюда необходимо ввести параметры валютного счета, оформить который рекомендовалось ранее. Пока эти действия не будут выполнены, загрузка платных программ в App Store не разрешается;
- **Payments and Financial Reports** — информация о выплатах. Минимальная сумма выплаты составляет 150 долларов, если программ продано на меньшую сумму, деньги остаются до следующего месяца;
- **Manage Users** — можно настраивать разрешения работы для нескольких пользователей с различными правами доступа, например Admin или Developer;

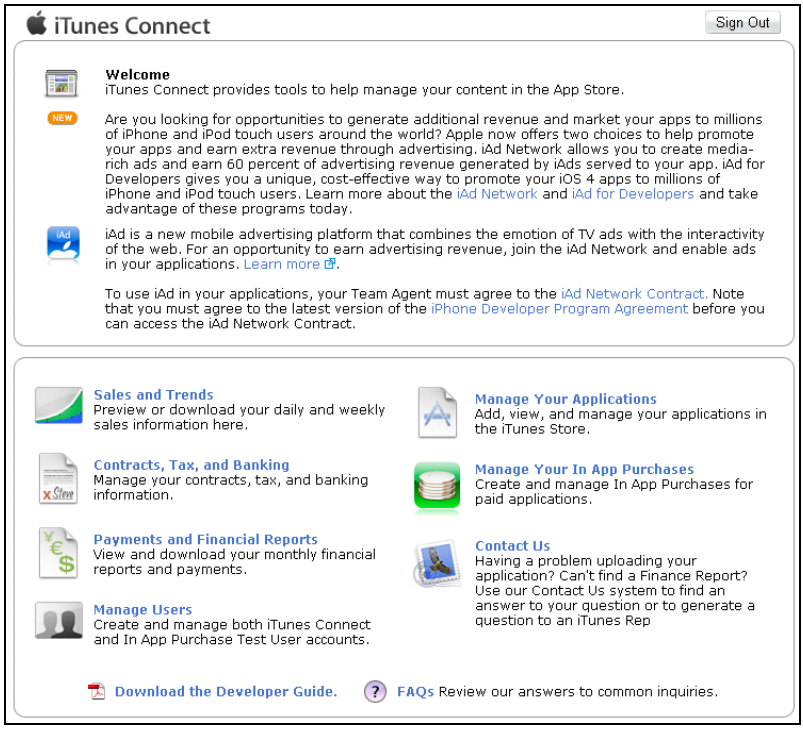


Рис. 9.10. Раздел **iTunes Connect**

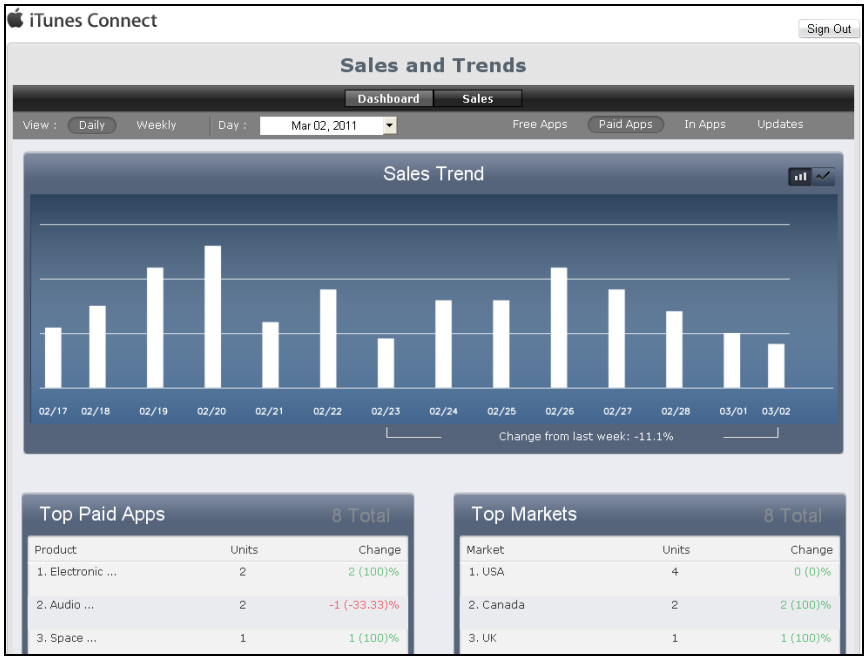


Рис. 9.11. Окно **Sales and Trends**

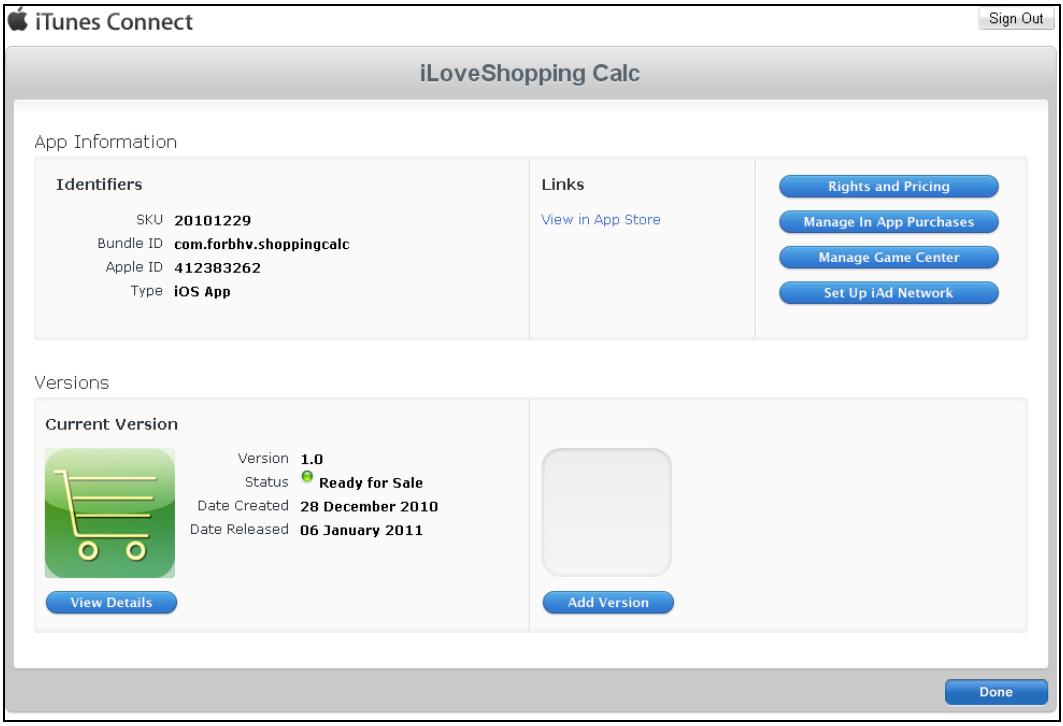


Рис. 9.12. Описание программы в разделе Manage Your Applications

- **Manage Your In App Purchases** — учетные записи для In App Purchases. Каждой записи соответствует тип покупки (разовый, периодический), стоимость и пр.;
- **Manage Your Applications** — список приложений, доступных для загрузки. На рис. 9.12 приведен пример приложения с параметрами:
  - **Rights and Pricing** — цена на программу фиксированная или может изменяться к определенной дате;
  - **Manage In App Purchases** — управление покупками из программы;
  - **Manage Game Center** — управление ресурсами для многопользовательских игр;
  - **Set Up iAd Network** — управление отображением рекламы в приложении;
  - **View Details** — просмотреть текущую информацию о приложении;
  - **Add Version** — загрузить новую версию. Версия ставится в очередь на рассмотрение в App Store, и после того как программа будет принята, старая версия будет замещена новой.

Как уже говорилось, для оценки продаж гораздо более удобна программа iTunes Connect Mobile, использовать которую можно на iPhone и iPad.

Требование к программам, продаваемым через App Store

До сих пор мы изучали коды программ, которые были вполне работоспособны. Однако для размещения в App Store этого недостаточно. Для того чтобы программа была готова к продаже, нужно выполнить ряд условий, которые описаны в уже упомянутом ранее документе "iOS Human Interface Guidelines". Программа также должна иметь минимально необходимый набор ресурсов, без которых ее размещение невозможно. Рассмотрим их более подробно.

Значок приложения (Application icon)

Рассмотрим значки приложений на рис. 9.13.

Их унифицированный вид (закругление краев и блик сверху) автоматически обеспечивается операционной системой. Для корректного отображения на разных устройствах программа должна иметь в ресурсах значки трех типов:

- 57×57 — значок для iPhone;
- 114×114 — значок для iPhone с дисплеем высокого разрешения;
- 72×72 — значок для iPad.

Рис. 9.13. "Рабочий стол" iPhone (фрагмент)

Даже если приложение создано исключительно для iPhone, в программе нужно иметь значки всех трех типов. Ведь приложение для iPhone может быть загружено и на iPad.

Для того чтобы добавить в программу необходимые значки, необходимо сделать следующее.

1. Подготовить файлы требуемого размера.

> ***Важное примечание***
>
> Закругление значков и блик делать не нужно, этот эффект формируется автоматически. Но при большом желании автоматическую генерацию эффекта можно отключить, если добавить в файл plist флаг `UIPrerenderedIcon` и установить его значение в `TRUE`. Это может быть полезно, например, если значки уже с закруглениями.

2. Добавить в plist-файл ключ с именами и описаниями новых значков в XCode или нужные строки с помощью текстового редактора, как показано в листинге 9.1.

Листинг 9.1. Строки описания значков в plist-файле

```
<key>CFBundleIconFile</key>
<string>Icon-iPhone.png</string>
<key>CFBundleIconFiles</key>
<array>
        <string>Icon-iPad.png</string>
        <string>Icon-iPhone.png</string>
        <string>Icon-iPhoneHD.png</string>
</array>
```

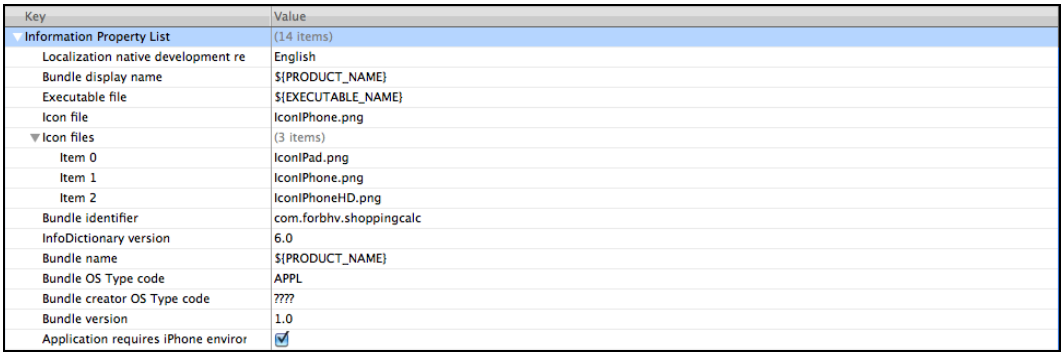


Рис. 9.14. Отображение значков в файле plist

После того как значки созданы, их необходимо добавить в программу с помощью соответствующей команды меню XCode (рис. 9.14). Для проверки достаточно удалить старую версию программы из симулятора и загрузить новую, значок на экране должен обновиться.

Изображение для загрузки (Launch image)

Эти изображения отображаются во время запуска программы. Простейший пример — черный экран с надписью "Loading…" в центре, который часто встречается в компьютерных играх. Для неигровых программ рекомендуется создавать Launch image по образцу интерфейса программы (рис. 9.15): сначала появляется фоновый рисунок, затем (через 1−2 секунды), когда приложение загрузится, поверх рисунка появляется текст.

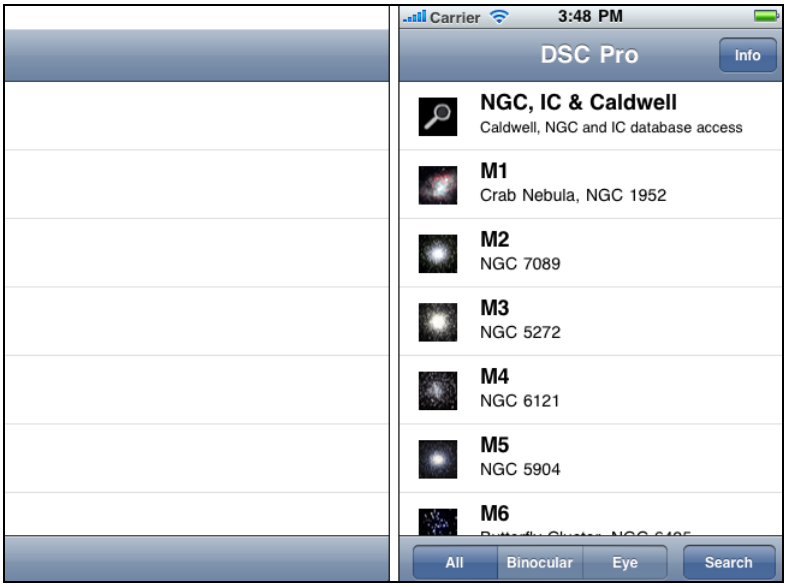


Рис. 9.15. Launch image (слева) и интерфейс приложения (справа)

Очевидно, что для разных устройств форматы изображений должны быть различны:

- для iPhone и iPod размер изображения должен составлять 320×480 пикселов, этот размер соответствует разрешению экрана;
- для iPhone и iPod с экранами высокого разрешения размер изображения должен составлять 640×960 пикселов;
- для iPad размер изображения составляет… вовсе не 1024×768, как можно было бы подумать. Тут все немного сложнее. Разработчик должен учесть вертикальное и горизонтальное положения экрана, причем без верхней панели статуса — два изображения размером 768×1004 и 1024×748 соответственно.

Изображения должны иметь специальные имена:

- Default.png — файл для iPhone 320×480;
- Default@2x.png — файл для iPhone с дисплеем Retina 640×960;
- Default-Portrait.png — файл для iPad 768×1004;
- Default-Landscape.png — файл для iPad 1024×748.

Все эти файлы необходимо скопировать в ресурсы программы и добавить с помощью XCode.

Значок App Store (App Store icon)

Внешний вид значка приложения в App Store можно видеть на рис. 9.16.

Это в точности такой же значок, как и в программе, только значительно большего размера — 512×512 пикселов. В отличие от предыдущих значков, эту картинку размещать внутри программы не нужно. Изображение необходимо сохранить

в виде PNG-файла и загрузить на сервер App Store во время размещения программы (без закругления, оно автоматически формируется при загрузке программы на сервер Apple).

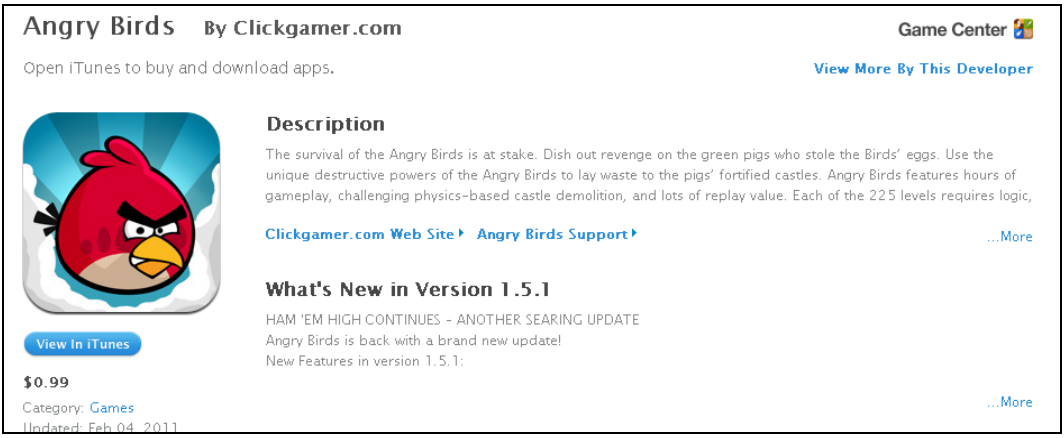


Рис. 9.16. Значок в описании приложения

Идентификатор программы (Bundle identifier)

Идентификатор программы должен быть уникальным, и после публикации программы изменить его уже нельзя. Для создания идентификатора рекомендуется использовать обратное написание адреса сайта программы, например `com.mysite.myprog`. Созданный нами идентификатор в plist-файле можно видеть на рис. 9.14.

Этот же идентификатор необходимо ввести на сайте при генерации сертификатов и вводе описания программы, как будет показано далее.

Описание программы и ключевые слова

Для публикации программы необходимо подготовить ее описание на английском языке и список ключевых слов. В дальнейшем можно добавить через сайт описание и на других языках.

Возрастные ограничения

Разработчику предлагается довольно внушительный список того, что может или не может выводить приложение (рис. 9.17).

Если никакая из категорий не установлена, программа автоматически получает возрастной рейтинг "4+". По набору значений в категориях (в качестве примера можно привести жанр стрип-покеров) возрастной рейтинг программы будет автоматически пересчитан.

Apple Content Descriptions	None	Infrequent/Mild	Frequent/Intense
Cartoon or Fantasy Violence	◉	○	○
Realistic Violence	◉	○	○
Sexual Content or Nudity	◉	○	○
Profanity or Crude Humor	◉	○	○
Alcohol, Tobacco, or Drug Use or References	◉	○	○
Mature/Suggestive Themes	◉	○	○
Simulated Gambling	◉	○	○
Horror/Fear Themes	◉	○	○
Prolonged Graphic or Sadistic Realistic Violence	◉	○	○
Graphic Sexual Content and Nudity	◉	○	○

Рис. 9.17. Список возможных возрастных ограничений, предлагаемый Apple

На этом мы закончим с "теоретической частью" и перейдем к практике — создадим платное приложение и опубликуем его в App Store.

Глава 10

Создание и разработка программы для App Store

Настало время выпустить настоящее коммерческое приложение.

Постановка задачи

Для начала стоит решить, что же именно будет разрабатываться. В книгах пишется о маркетинговых исследованиях рынка, но обычный разработчик такой возможности лишен. Постановка задачи — это, скорее, вопрос интуиции и, безусловно, личных предпочтений. Описывалось немало случаев, когда разработчик делал программу для себя, в качестве хобби, и затем она приносила коммерческий успех.

Можно присмотреться к обзорам и рейтингам продаваемых программ (естественно, речь не идет о простом клонировании). Поэтому сделать что-то свое — может быть, более перспективно, есть шанс занять узкоспециализированную нишу и иметь определенную прибыль.

Есть еще третий путь — это разработка программ, которые всегда нужны: разнообразные часы, будильники, органайзеры, менеджеры закачек и т. п. Если программа сделана качественно и красиво, она будет продаваться (с другой стороны, так же легко она может затеряться среди конкурентов). Автор может поделиться небольшой коммерческой тайной: программа, выводящая электронные часы и прогноз погоды, за 3 месяца принесла 130 закачек и соответственно около 90 долларов прибыли. С точки зрения разработчика, это позволит почувствовать свои силы, и даже небольшая прибыль придаст уверенности в себе.

Впрочем, наша задача на данный момент проще — это создание учебного проекта. Поэтому нужно что-то простое, но наглядное и востребованное, например, калькулятор. Обычных калькуляторов в App Store много, поэтому в программе будет дополнительная функция: это будет калькулятор для походов в магазин, позволяющий не только делать подсчеты, но и выводить список покупок и суммарную стоимость.

Разработка программы

Создадим новый проект с названием Shopping. Настоящее имя программе дадим позже, в момент загрузки в App Store.

Для походов в магазин с программой нужно иметь экран, стилизованный под жидкокристаллический индикатор калькулятора, клавиши ввода цифр и список покупок.

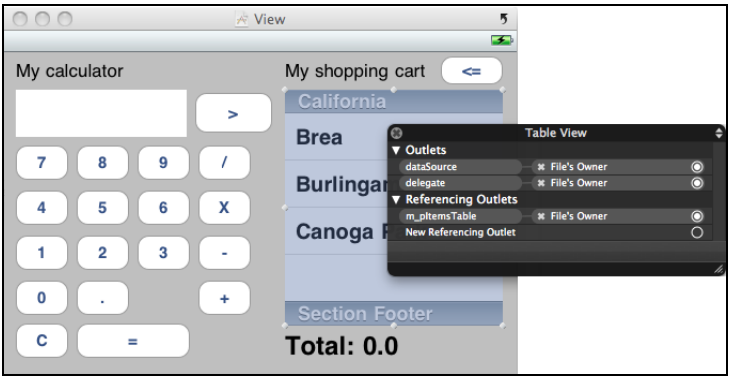


Рис. 10.1. Элементы интерфейса программы

Таким образом, в основном окне программы нам нужны объекты (рис. 10.1):

- элемент управления, наследуемый от `UIView`;
- набор кнопок `UIButton` калькулятора;
- таблица `UITableView` для вывода списка.

Установим связи, создав предварительно функции обработчиков кнопок цифр, арифметических операций, добавления и удаления из списка, используем в классе протоколы `UITableViewDelegate` и `UITableViewDataSource`. Таблица связана с элементом `m_pItemsTable`, имеющим тип `UITableView`.

Рассмотрим последовательно все компоненты программы.

Список покупок

Для использования списка необходимо создать класс, обеспечивающий хранение необходимых данных таблицы, из двух полей — наименования и цены покупки.

Код показан в листинге 10.1.

Листинг 10.1. Класс для хранения элемента списка покупок

```
// Data object
@interface DataItem : NSObject
{
        float fPrice;
        NSString *sDescription;
}

- (id) initWithData: (float)price: (NSString*)description;

@property float fPrice;
@property(nonatomic, copy)    NSString              *sDescription;

@end
```

```
@implementation DataItem
@synthesize fPrice;
@synthesize sDescription;

- (id) initWithData: (float)price: (NSString*)description
{
        self = [super init];

        self.fPrice = price;
        self.sDescription = description;

        return self;
}

- (void)dealloc
{
        [sDescription release];

    [super dealloc];
}

@end
```

Для хранения элементов используем массив `NSMutableArray`. При нажатии кнопки **Add** новый элемент будет добавляться в список, как показано в листинге 10.2.

Листинг 10.2. Добавление элементов

```
DataItem *item = [[[DataItem alloc] initWithData: val: text]
                                                    autorelease];

// Add value
[m_pShoppingData insertObject:item atIndex:0];
[m_pItemsTable   reloadData];
// Scroll to top
[m_pItemsTable scrollRectToVisible:CGRectMake(0, 0, 1, 1) animated:YES];

// Recalculate
[self CalculateSum];
```

Переменные `val` и `text` в коде не показаны, эти данные будут браться из калькулятора, который мы создадим далее. Пока что их можно заменить любыми временными переменными, например написать:

```
NSString *text = @"abcd"; float val = 10.0f;
```

Код инициализации и отображения таблицы показан в листинге 10.3.

Листинг 10.3. Вывод списка покупок

```
- (NSInteger)numberOfSectionsInTableView:(UITableView *)tableView
{
   return 1;
}

- (NSInteger)tableView:(UITableView *)tableView
                     numberOfRowsInSection:(NSInteger)section
{
   return [m_pShoppingData count];
}

- (UITableViewCell *)tableView:(UITableView *)tableView
                       cellForRowAtIndexPath:(NSIndexPath *)indexPath
{
   static NSString *MyIdentifier = @"IDDQD"; // any ID

   UITableViewCell *cell = [m_pItemsTable
                    dequeueReusableCellWithIdentifier:MyIdentifier];

   // If no cell is available, create a new one using
   // the given identifier.
   if (cell == nil)
   {
      // Use the default cell style.
      cell = [[[UITableViewCell alloc]
                          initWithStyle:UITableViewCellStyleDefault
                          reuseIdentifier:MyIdentifier] autorelease];
   }

   DataItem *item = [m_pShoppingData objectAtIndex:indexPath.row];
   NSString *str;
   if (item.sDescription.length != 0)
      str = [NSString stringWithFormat:@"%@: %.2f",
                                              item.sDescription,
                                              item.fPrice];
       else
      str = [NSString stringWithFormat:@"%.2f", item.fPrice];


   // Set text
   cell.textLabel.text = str;

   return cell;
```

```
}

- (NSIndexPath *)tableView:(UITableView *)tableView
                      willSelectRowAtIndexPath:(NSIndexPath *)indexPath
{
   return indexPath;
}

- (void)tableView:(UITableView *)tableView
                        didSelectRowAtIndexPath:(NSIndexPath *)indexPath
{
   NSNumber *value = [m_pShoppingData objectAtIndex:indexPath.row];

   UIAlertView *simpleAlert = [[UIAlertView alloc]
                      initWithTitle:@"Shopping cart"
                      message:@"Delete this item from the list?"
                      delegate:self cancelButtonTitle:@"Yes"
                      otherButtonTitles:@"No", nil];
   simpleAlert.tag = ALERT_DEL;
   [simpleAlert show];
   [simpleAlert release];
}

- (void)alertView:(UIAlertView *)alertView
                 clickedButtonAtIndex:(NSInteger)buttonIndex
{
   switch (alertView.tag)
   {
   case ALERT_ADD:
      // код будет добавлен далее
      break;
   case ALERT_DEL:
      {
         NSIndexPath *t_sel = [m_pItemsTable indexPathForSelectedRow];
         if (t_sel != nil)
         {
            int n_sel = [t_sel row];        // Remember selection
            [m_pItemsTable deselectRowAtIndexPath:t_sel animated:YES];

            if (buttonIndex == 0) // Yes button
            {
               if (n_sel >= 0 && n_sel < [m_pShoppingData count])
               {
                  m_pShoppingData removeObjectAtIndex:n_sel];
                  [m_pItemsTable   reloadData];
```

```
                    [self CalculateSum];

                    //Save data
                    [self SaveData];
                }
            }
        }
        break;
    }
}

- (void) CalculateSum
{
    float f_sum = 0.0f;
    for(DataItem *item in m_pShoppingData)
    {
        f_sum += item.fPrice;
    }

    m_pSumResults.text = [NSString stringWithFormat:@"Total: %.2f",
                                                    f_sum];
}
```

Мы заполняем массив `m_pShoppingData` данных таблицы. Щелчок по элементу таблицы служит для его удаления, а чтобы пользователь не удалил элемент при случайном нажатии, выводится соответствующий запрос.

Можно проверить функционирование таблицы покупок (рис. 10.2).

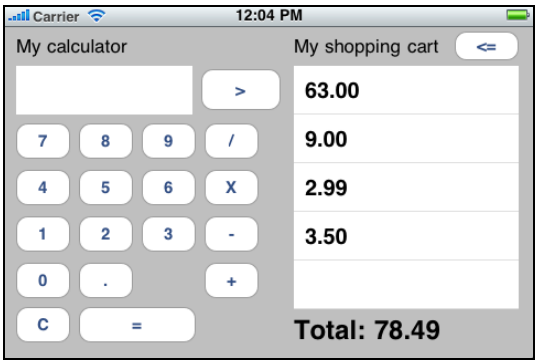


Рис. 10.2. Тестирование списка покупок

Теперь нам необходим калькулятор, но чтобы его тестировать, вначале сделаем компонент вывода цифровых данных.

Цифровой индикатор

Нас не устроит вывод статического текста, приложение должно быть красивым и действительно напоминать калькулятор. Поэтому мы сделаем цифры на базе "настоящего" сегментного индикатора (рис. 10.3).

Рис. 10.3. Изображения цифр индикатора

Для вывода создадим класс, наследуемый от `UIView` (листинг 10.4).

Листинг 10.4. Вывод цифр в стиле цифрового индикатора

Файл DigitsView.h:

```
@interface DigitsView : UIView
{
       NSString *m_sDigitData;
       BOOL     m_bUseRound;
}

@property (nonatomic, copy) NSString *m_sDigitData;

- (void) addDigit: (int)nDigit;
- (void) resetDigit;
- (void) setDigit: (float)fDigit;
@end
```

Файл DigitsView.m:

```
@implementation DigitsView
@synthesize m_sDigitData;

- (void) addDigit: (int)nDigit
{
    if (nDigit >= 0 && nDigit <= 9)
        self.m_sDigitData = [NSString stringWithFormat:@"%@%d",
                                        self.m_sDigitData, nDigit];
    if (nDigit == DOT)
        self.m_sDigitData = [NSString stringWithFormat:@"%@.",
                                        self.m_sDigitData];

    m_bUseRound = FALSE;
    [self setNeedsDisplay];
}
```

```
- (void) resetDigit
{
    self.m_sDigitData = @"";

    m_bUseRound = FALSE;
    [self setNeedsDisplay];
}

- (void) setDigit: (float)fDigit
{
    self.m_sDigitData = [NSString stringWithFormat:@"%f", fDigit];

    m_bUseRound = TRUE;
    [self setNeedsDisplay];
}

- (void)drawRect:(CGRect)rect
{
    NSString *s_digits = @"0";
    if (self.m_sDigitData.length != 0)
        s_digits = self.m_sDigitData;

    // Round in 2 digits
    if (m_bUseRound)
    {
        float val = [s_digits floatValue];
        if (val == (int)(val))
            s_digits = [NSString stringWithFormat:@"%d", (int)(val)];
                else
            s_digits = [NSString stringWithFormat:@"%.2f", val];
    }

    const int IMG_W = 20, IMG_H = 40, DX = 2, N_DIGITS = 7;
    const int   CX = (rect.size.width - (IMG_W + DX)*N_DIGITS)/2,
                CY = (rect.size.height - IMG_H)/2;

    // Fill rect
    CGContextRef dc = UIGraphicsGetCurrentContext();
    CGContextSetRGBFillColor(dc, 159.0/255.0, 168.0/255.0, 154.0/255.0,
                                 1.0);
    CGContextFillRect(dc, rect);

    // Cut string if too long
    if ([s_digits length] >= N_DIGITS)
```

```
    s_digits = [s_digits substringToIndex:N_DIGITS];

// 7-digits indicator: init
unichar digits[N_DIGITS];
for(int i=0; i<N_DIGITS; i++)
    digits[i] = ' ';
// Check: number too big
if (fabs([s_digits floatValue]) > 999999)
{
    digits[0] = digits[1] = digits[2] = '-';
} else
{
    for(int i=0; i<s_digits.length; i++)
        digits[N_DIGITS - s_digits.length + i] = [s_digits
                                            characterAtIndex:i];
}


// 7-digits indicator: draw
int x_pos = CX;
for(int i=0; i<N_DIGITS; i++)
{
    switch (digits[i])
    {
    case '0':
        [[UIImage imageNamed:@"S1DigitSmall00.png"]
                        drawAtPoint:CGPointMake(x_pos, CY)];
        break;
    case '1':
       [[UIImage imageNamed:@"S1DigitSmall01.png"]
                        drawAtPoint:CGPointMake(x_pos, CY)];
        break;
    case '2':
       [[UIImage imageNamed:@"S1DigitSmall02.png"]
                        drawAtPoint:CGPointMake(x_pos, CY)];
       break;
    case '3':
       [[UIImage imageNamed:@"S1DigitSmall03.png"]
                        drawAtPoint:CGPointMake(x_pos, CY)];
       break;
    case '4':
       [[UIImage imageNamed:@"S1DigitSmall04.png"]
                        drawAtPoint:CGPointMake(x_pos, CY)];
        break;
    case '5':
```

```
            [[UIImage imageNamed:@"S1DigitSmall05.png"]
                                drawAtPoint:CGPointMake(x_pos, CY)];
            break;
        case '6':
            [[UIImage imageNamed:@"S1DigitSmall06.png"]
                                drawAtPoint:CGPointMake(x_pos, CY)];
            break;
        case '7':
          [[UIImage imageNamed:@"S1DigitSmall07.png"]
                              drawAtPoint:CGPointMake(x_pos, CY)];
           break;
        case '8':
            [[UIImage imageNamed:@"S1DigitSmall08.png"]
                                drawAtPoint:CGPointMake(x_pos, CY)];
            break;
        case '9':
            [[UIImage imageNamed:@"S1DigitSmall09.png"]
                                drawAtPoint:CGPointMake(x_pos, CY)];
            break;
        case ' ':
            [[UIImage imageNamed:@"S1DigitSmallEE.png"]
                                drawAtPoint:CGPointMake(x_pos, CY)];
           break;
        case '-':
            [[UIImage imageNamed:@"S1DigitSmallMinus.png"]
                                drawAtPoint:CGPointMake(x_pos, CY)];
           break;
        case '.':
            [[UIImage imageNamed:@"S1DigitSmallDot.png"]
                                drawAtPoint:CGPointMake(x_pos, CY)];
           break;
        }

        x_pos += IMG_W + DX;
    }
}

- (void)dealloc
{
    [m_sDigitData release];

    [super dealloc];
}

@end
```

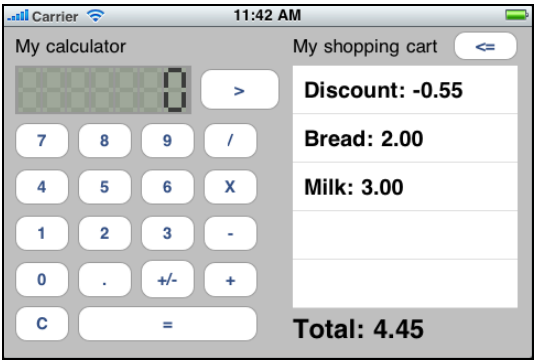


Рис. 10.4. Использование индикатора в программе

Когда пользователь нажимает клавишу на калькуляторе, мы добавляем цифру в строке, используя для этого функцию `addDigit`. Далее выводим строку посимвольно. Если выводимое число является результатом вычислений, используется функция `setDigit`, которая активирует переменную `m_bUseRound` для округления значений. Поскольку калькулятор нужен для расчетов в магазине, двух цифр после запятой вполне достаточно (рис. 10.4).

В программу была добавлена кнопка +/−, позволяющая вводить отрицательные значения скидок, предоставляемых при покупке.

Калькулятор

Теперь, когда цифры удобно выводятся на экран, можно приступить к созданию и отладке калькулятора. Весь код здесь приводить не будем, остановимся лишь на основных фрагментах.

В калькуляторе имеется три вида кнопок:

- цифровые кнопки;
- кнопки арифметических действий (при их нажатии введенное значение и тип операции запоминаются, и калькулятор переходит в режим ввода второго числа);
- кнопка вычисления результата.

На практике при вводе второго числа пользователь сразу может нажать еще раз кнопку арифметического действия и перейти к вводу третьего. Но принципиально кода это не меняет (листинг 10.5). Поскольку окно калькулятора является основным окном программы, необходимые переменные и функции добавлены в этот же класс. (Смешивание логики и графического интерфейса в одном классе нежелательно, но для относительно несложного проекта вполне допустимо.)

Листинг 10.5. Основной класс приложения

Файл ShoppingViewController.h:

```
@interface ShoppingViewController: UIViewController<UITableViewDelegate,
                                                    UITableViewDataSource,
                                                    UIAlertViewDelegate>
```

```
{
        // Calc data
        int m_nOperationStage; // 0 - first, 1 - second
        int m_nOperationType;
        int m_nButtonPressedType;
        NSString *m_sDigitFirst, *m_sDigitSecond;
}
- (IBAction) onDigit0:(id)sender;
…
- (IBAction) onDigit9:(id)sender;
- (IBAction) onDigitClear:(id)sender;
- (IBAction) onOperationPlus:(id)sender;
- (IBAction) onOperationMinus:(id)sender;
- (IBAction) onOperationMul:(id)sender;
- (IBAction) onOperationDiv:(id)sender;
- (IBAction) onOperationResult:(id)sender;
- (IBAction) onOperationResult:(id)sender;
- (void) AddDigit: (int)nDigit;
- (void) CalculateResult;
- (void) NextOperation;
@end
```

Файл ShoppingViewController.m:

```
// Calc buttons
#define DOT            -1
#define OP_NONE         0
#define OP_PLUS         1
#define OP_MINUS        2
#define OP_MUL          3
#define OP_DIV          4
#define OP_ADDTOCART    5
#define BTN_RES         10
// Current operation
#define FIRST           0
#define SECOND          1

@implementation ShoppingViewController
@synthesize m_sDigitFirst;
@synthesize m_sDigitSecond;

- (IBAction) onDigit0:(id)sender
{
    [self AddDigit:0];
}

…
```

```
- (IBAction) onDigit9:(id)sender
{
     [self AddDigit:9];
}

- (void) AddDigit: (int)nDigit
{
    // new operation after '=' or 'add' was pressed
    if (m_nButtonPressedType == BTN_RES ||
        m_nButtonPressedType == OP_ADDTOCART)
    {
         [m_pDigitsView resetDigit];
         m_nOperationType = m_nButtonPressedType = OP_NONE;
         m_nOperationStage = FIRST;
    }
    // reset digit
    if (m_nButtonPressedType == OP_PLUS ||
        m_nButtonPressedType == OP_MINUS ||
        m_nButtonPressedType == OP_MUL ||
        m_nButtonPressedType == OP_DIV)
    {
         [m_pDigitsView resetDigit];
         m_nButtonPressedType = OP_NONE;
    }

    [m_pDigitsView addDigit:nDigit];

    if (m_nOperationStage == FIRST)
        self.m_sDigitFirst = m_pDigitsView.m_sDigitData;
    if (m_nOperationStage == SECOND)
        self.m_sDigitSecond = m_pDigitsView.m_sDigitData;
}

- (IBAction) onOperationPlus:(id)sender
{
    if (m_nButtonPressedType == OP_PLUS) return;

    [self NextOperation];
    m_nOperationType = m_nButtonPressedType = OP_PLUS;
}

- (void) NextOperation
{
    if (m_nOperationStage == FIRST)
    {
```

```
        m_nOperationStage = SECOND;
    } else
    {
        [self CalculateResult];
        self.m_sDigitFirst = m_pDigitsView.m_sDigitData;
        m_nOperationStage = SECOND;
    }
}

- (IBAction) onOperationResult:(id)sender
{
    if (m_nOperationType == OP_NONE ||
        m_nButtonPressedType == OP_ADDTOCART) return;

    [self CalculateResult];

    self.m_sDigitFirst = m_pDigitsView.m_sDigitData;
    m_nOperationStage = FIRST; // ready to second operation

    m_nButtonPressedType = BTN_RES;
}

- (void) CalculateResult
{
    float val1, val2, result = 0.0f;
    val1 = [self.m_sDigitFirst  floatValue];
    val2 = [self.m_sDigitSecond floatValue];

    switch (m_nOperationType)
    {
    case OP_PLUS:
        result = val1 + val2;
        break;
    case OP_MINUS:
        result = val1 - val2;
        break;
    case OP_MUL:
        result = val1*val2;
        break;
    case OP_DIV:
        result = fabs(val2) > 1E-5?val1/val2:0.0f;
        break;
    }

    [m_pDigitsView setDigit:result];
}
```

Наш калькулятор содержит основные арифметические операции, и для походов в магазин этого, по идее, вполне достаточно.

Теперь соединим части программы и добавим результат в список покупок.

Добавление покупок в таблицу

Вернемся к рис. 10.4. Как можно видеть, нам необходимо хранить название покупки. Несмотря на кажущуюся простоту, здесь есть один "подводный камень". Проблема заключается в том, что стандартного компонента для ввода строки в виде всплывающего окна в iOS нет. Но в классе `UIAlertView` имеется недокументированная функция `addTextFieldWithValue`, как раз предназначенная для решения этой задачи (листинг 10.6).

Листинг 10.6. Вывод окна добавления покупок

```
- (IBAction) onButtonAdd:(id)sender
{
    float val = [m_pDigitsView.m_sDigitData floatValue];

    NSString *price = [NSString stringWithFormat:@"Price: %.2f", val];

    UIAlertView *simpleAlert = [[UIAlertView alloc] initWithTitle:@"Add
                                                          item to cart"
                                message:price
                                delegate:self cancelButtonTitle:@"Yes"
                                otherButtonTitles:@"No",nil];
    simpleAlert.tag = ALERT_ADD;
    [simpleAlert addTextFieldWithValue:@"" label:nil];
    [simpleAlert show];
    [simpleAlert release];
}

- (void)alertView:(UIAlertView *)alertView
                          clickedButtonAtIndex:(NSInteger)buttonIndex
{
    switch (alertView.tag)
    {
    case ALERT_ADD:
        if (buttonIndex == 0) // Yes button
        {
            float val = [m_pDigitsView.m_sDigitData floatValue];
            NSString *text = [alertView textField].text;

            DataItem *item = [[[DataItem alloc] initWithData: val: text]
                                                          autorelease];
```

```
            // Add value
            [m_pShoppingData insertObject:item atIndex:0];
            [m_pItemsTable   reloadData];
            // Scroll to top
            [m_pItemsTable scrollRectToVisible:CGRectMake(0, 0, 1, 1)
                          animated:YES];

            // Recalculate
            [self CalculateSum];

            //Save data
            [self SaveData];
        }
        break;
    case ALERT_DEL:
        // Этот код рассматривался ранее
        break;
    }
}
```

Мы добавили текстовое поле с помощью функции `addTextFieldWithValue`, затем получили значение введенного текста `[alertView textField].text`.

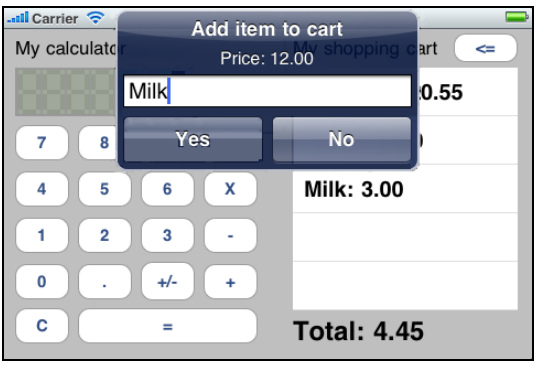


Рис. 10.5. Окно добавления покупок

Итак, наша программа готова (рис. 10.5), но еще не очень похожа на коммерческое приложение. Во-первых, требуется сохранять список покупок при нажатии кнопки **Home** или переключении в другое приложение (в операционной системе iOS 4.0 поддерживается многозадачность, и при переключении программы данные не пропадают). Кроме того, необходимо доработать интерфейс программы, сделать фон и кнопки более похожими на "настоящий" калькулятор.

Запись списка покупок при выходе

Чтение и загрузку настроек программы мы рассматривали ранее в соответствующих главах.

Листинг 10.7. Сохранение и открытие списка покупок

```
- (void) LoadData
{
    // Load array
    NSNumber *count = [self GetProfileInt:@"ItemsCount"];
    if (count == nil) return;

    [m_pShoppingData removeAllObjects];
    for (int i=0; i<[count intValue]; i++)
    {
        NSNumber *price = [self GetProfileFloat:i :@"ItemPrice"];
        NSString *descr = [self GetProfileString:i :@"ItemDescription"];

        if (price == nil || descr == nil) continue;

        DataItem *item = [[[DataItem alloc] initWithData:
                                            [price floatValue]:
                                            descr]
                                            autorelease];

        // Add value
        [m_pShoppingData addObject:item];
    }
}

- (void) SaveData
{
    // Save array
    int count = m_pShoppingData.count;
    [self SetProfileInt: count :@"ItemsCount"];

    for(int i=0; i<count; i++)
    {
        DataItem *item = [m_pShoppingData objectAtIndex:i];
        if (item != nil)
        {
         [self SetProfileFloat:item.fPrice :i:@"ItemPrice"];
         [self SetProfileString:item.sDescription: i:@"ItemDescription"];
        }
    }
}
```

Код в листинге 10.7, возможно, не слишком эффективный (компактнее хранить данные в бинарном формате), зато он читабельный, а при таких объемах данных, как в этом приложении, экономия пары десятков байтов не сильно критична. Функции `GetProfileInt`, `SetProfileInt`, `SetProfileString` рассматривались ранее.

Для чтения списка мы вставим соответствующий вызов в функцию `viewDidLoad`, для записи воспользуемся функциями `onButtonAdd` и `onButtonRemove`, список будет сохраняться только тогда, когда пользователь его изменяет.

Дизайн

Операционная система iOS предоставляет большие возможности, позволяя использовать графические изображения практически для всех элементов управления. Для того чтобы сделать программу более красивой, нам необходимо создать фоновый рисунок для главного окна приложения и отдельные изображения для кнопок.

Фоновое изображение показано на рис. 10.6. Этот рисунок будет "подложкой", на которой будут размещены остальные элементы (рис. 10.7).

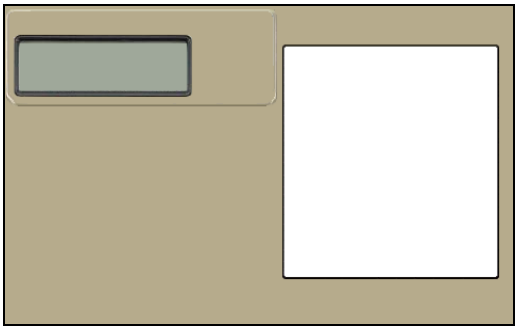


Рис. 10.6. Фоновый рисунок главного окна

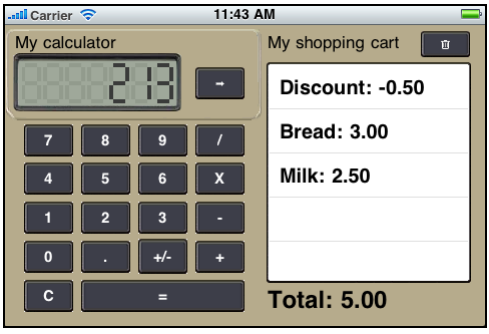


Рис. 10.7. Окончательный вариант программы

Это изображение было составлено из нескольких различных рамок и калькуляторов, которые удалось найти в сети Интернет. По личному опыту, приглашение дизайнера на разовую работу даже небольшого объема (3−5 картинок — фон, цифры, кнопки, значки) невыгодно. Поэтому приходится обходиться своими силами, что, впрочем, иногда интересно.

И наконец, необходимо указать поддерживаемые режимы поворота экрана с помощью plist-файла.

Наша программа может работать только в горизонтальной ориентации экрана. Необходимо задать параметр **Supported interface orientations**, как показано на рис. 10.8, и внести изменения в `shouldAutorotateToInterfaceOrientation`.

На этом мы закончим с программой и перейдем к подготовке размещения в App Store.

Key	Value
Information Property List	(14 items)
Localization native development re	English
Bundle display name	${PRODUCT_NAME}
Executable file	${EXECUTABLE_NAME}
Icon file	IconIPhone.png
Icon files	(3 items)
Item 0	IconIPad.png
Item 1	IconIPhone.png
Item 2	IconIPhoneHD.png
Bundle identifier	com.forbhv.shoppingcalc
InfoDictionary version	6.0
Bundle name	${PRODUCT_NAME}
Bundle OS Type code	APPL
Bundle creator OS Type code	????
Bundle version	1.0
Application requires iPhone enviror	☑
Main nib file base name	MainWindow
Supported interface orientations	(2 items)
Item 0	Landscape (left home button)
Item 1	Landscape (right home button)

Рис. 10.8. Plist-файл программы

Подготовка требуемых для App Store текстовых и графических материалов

Эта программа не имеет поддержки различных языков интерфейса, поэтому объем текстовой информации в ней минимален. Главное, что нужно сделать — записать уникальный идентификатор программы (в нашем примере он называется `com.forbhv.shoppingcalc`) в поле **Bundle identifier**.

Что касается локализации, по личному мнению автора, затраты времени на поддержку различных языков себя не оправдывают. По крайней мере, для рассматриваемой программы локализацию мы делать не будем.

Описание программы

Вначале было составлено описание на русском языке, которое выглядит так:

Программа Shopping Calc — это очень простой и удобный калькулятор для совершения покупок. С помощью программы можно не только произвести все необходимые вычисления, но и внести получившиеся результаты в список покупок. При добавлении или удалении каждой покупки Shopping Calc автоматически пересчитает общую сумму, которую нужно оплатить.

Этого вполне достаточно для потенциального покупателя, чтобы понять, что делает программа. Английская версия и ключевые слова приведены *в главе 11*, где рассмотрено непосредственное размещение программы в App Store.

Изображение для загрузки (Launch image)

Это изображение появляется во время старта программы. Согласно рекомендациям Apple, оно должно повторять фон интерфейса, но не иметь текстовых полей, чтобы у пользователя создавалась иллюзия "постепенного" появления на экране (рис. 10.10). Изображение размером 320×480 пикселов (программа стартует в вертикальной ориентации экрана) под именем Default.png хранится в ресурсах программы.

Для iPhone4 необходимо создать такой же файл вдвое большего разрешения (640×960) и сохранить его под именем Default@2x.png.

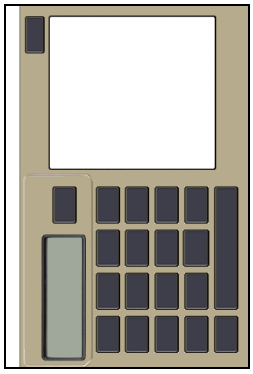


Рис. 10.10. Launch image

Значки программы

Значки программы для iPhone4, iPad и для "обычного" iPhone размерами 114×114, 72×72 и 57×57 пикселов соответственно показаны на рис. 10.9. Размер значка в App Store 512×512 пикселов.

Рис. 10.9. Значки программы

Еще раз стоит повторить, что закругления у значков делать не нужно, система добавляет их автоматически.

Размер скриншота должен составлять 480×320 пикселов для полноэкранных приложений и 480×300 для обычных без верхней панели. Apple позволяет сохранять до пяти скриншотов для каждой из платформ (iPhone и iPad), но для нашей программы вполне хватит и одного.

Теперь нашу программу можно считать законченной. Осталось лишь сгенерировать необходимые сертификаты и отправить приложение в App Store, чем мы и займемся в следующей главе.

Глава 11

Размещение программы в App Store

Создание сертификата программы

Публикация программы в App Store невозможна без наличия цифровых сертификатов, которыми подписывается приложение. Рассмотрим этот процесс подробнее.

Вначале необходимо создать Distribution certificate (сертификат распространения), являющийся базовым для всех приложений. Сгенерировать нужный сертификат следует в разделе **Cetrificates** портала Provisioning Portal, следуя инструкциям сайта App Store. Процедура создания состоит из нескольких шагов:

1. Запустить программу Keychain Access (Связка ключей), в ней выбрать пункт создания сертификата.
2. Сохранить сертификат на диск и запомнить папку, куда будет сохранен созданный файл.
3. Загрузить сохраненный файл на сайт — новый сертификат будет создан.
4. Сохранить созданный файл и добавить его в систему, дважды щелкнув на имени.

Еще раз отметим, что в этом разделе есть два вида сертификатов: один для разработки и отладки программ, другой — для распространения. Созданный сертификат состоит из "базового сертификата" (один для всех программ) и "сертификата приложения" для каждой конкретной программы.

На рис. 10.8 мы вносили в plist-файл идентификатор программы `com.forbhvshoppingcalc`. В разделе **App IDs** создадим новый идентификатор, как показано на рис. 11.1.

Введем название программы и идентификатор, точно как в программе. При нажатии кнопки **Submit** новая запись будет создана.

Откроем раздел **Provisioning | Distribution** и выберем пункт создания нового сертификата (рис. 11.2).

Здесь необходимо отметить следующие параметры:

- **Distribution Method** — метод распространения либо **App Store** (нужный нам вариант), либо **Ad Hoc** (распространение для списка конкретных устройств, обычно с целью тестирования);
- **Profile Name** — имя, которое будет отображаться для нашего сертификата;
- **App ID** — идентификатор программы.

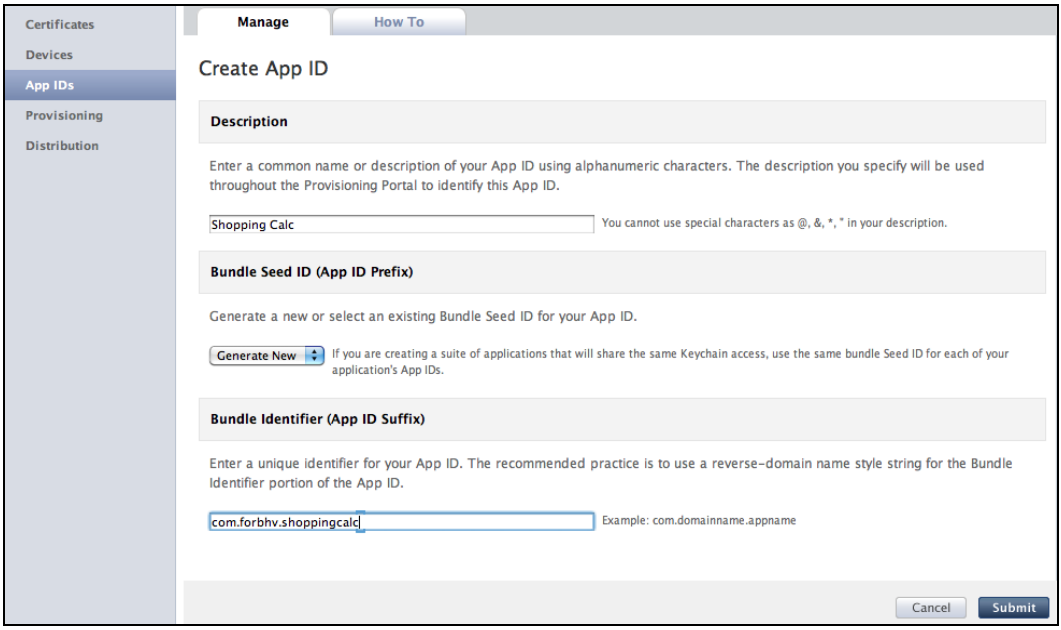


Рис. 11.1. Создание App ID

Рис. 11.2. Создание сертификата

По нажатию кнопки **Submit** создается новый сертификат, который также необходимо загрузить и дважды щелкнуть.

Сертификаты образуют пару ключей, необходимых для подписания программы. Если базовый сертификат будет удален, все сертификаты программ также необхо-

димо будет сформировать заново (старые не будут работать). Кроме того, если изменить идентификатор программы, при компиляции XCode сообщит, что не может подписать программу этим сертификатом.

Компиляция программы

После того как сертификаты созданы и сохранены на компьютере, необходимо скомпилировать отдельную версию программы для публикации. Есть два вида сертификатов: один для отладки (development), другой — для распространения (distribution). По умолчанию в XCode используется режим отладки, но в App Store такую программу распространять нельзя. Для компиляции заключительной версии необходимо выполнить следующие действия.

1. Открыть свойства проекта в XCode.
2. В разделе **Code Signing** выбрать раздел **Code Signing Identity** и в нем выбрать необходимый distribution-сертификат, как показано на рис. 11.3. Также рекомендуется проверить остальные свойства проекта (бывает, что настройки для версий Debug и Release отличаются). Например, версия поддерживаемой операционной системы по умолчанию часто максимальная, что ограничивает круг потенциальных клиентов.

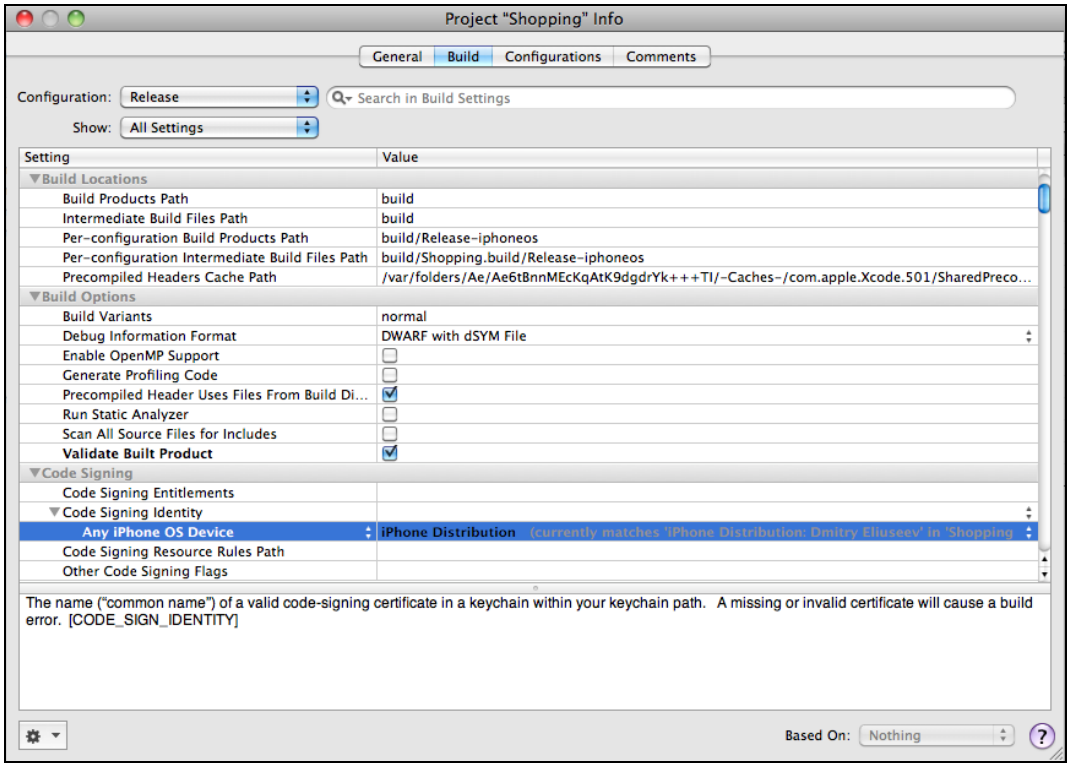


Рис. 11.3. Настройка свойств проекта для компиляции приложения

3. Удалить файлы предыдущих компиляций и скомпилировать новый проект. После этого в папке проекта появится каталог build\Release-iphoneos с файлом, имеющим расширение app. Его необходимо сохранить в виде zip-файла посредством контекстного меню программы Finder. Именно этот файл мы и будем отправлять в App Store. Итак, наше приложение полностью создано, подписано и готово к отправке.

Загрузка и размещение программы

К данному моменту мы имеем набор:

- архив с программой, подписанной необходимым сертификатом;
- значок для App Store;
- описание программы.

Этого достаточно для того, чтобы выкладывать программу в App Store. Для работы с приложениями служит портал iTunes Connect в панели управления учетными записями. Выбираем пункт **Manage Your Applications** (управление приложениями) и в появившемся окне выбираем пункт **Add New App**. Окно ввода данных о приложении показано на рис. 11.4.

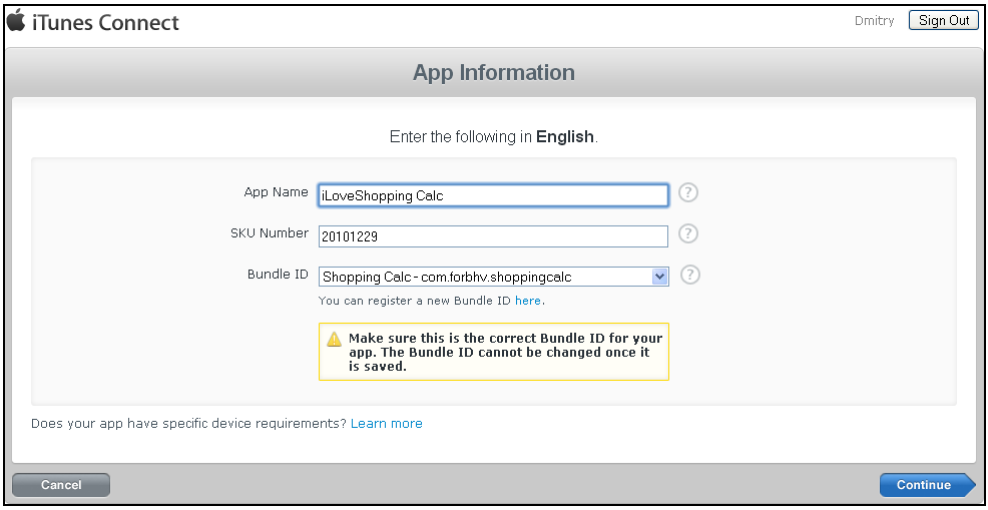


Рис. 11.4. Добавление новой программы

Вначале нужно заполнить поле **App Name** (Имя приложения). Желательно, чтобы имя было простое и легко запоминающееся. Но в App Store существует более 350 000 программ, и многие популярные имена, естественно, уже заняты. (В нашем примере было выбрано название iLoveShopping, поскольку оно было свободно.)

В поле **SKU Number** нужно ввести уникальное число, которое, как написано в документации, позволяет идентифицировать приложение среди остальных. Для того чтобы не путаться, записываем дату выпуска приложения, например `20101229`

для приведенного примера. **Bundle ID** — это уже известное нам поле идентификатора приложения.

Далее необходимо указать дату выпуска приложения и его стоимость, как показано на рис. 11.5.

Примечание

Срок рассмотрения программы обычно составляет от 5 до 10 дней, но описывались случаи, когда задержка оказывалась и большей.

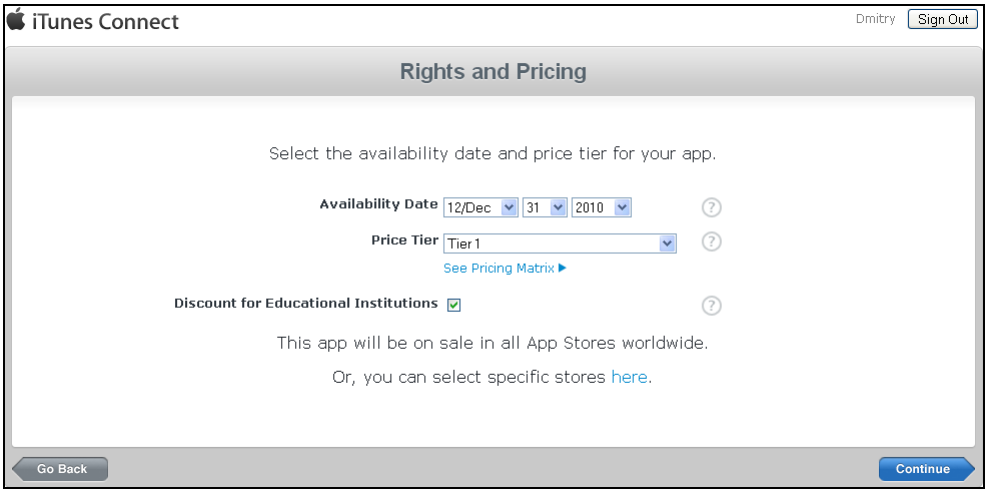


Рис. 11.5. Добавление новой программы, указание стоимости

Пользователь может указать цену из списка возможных вариантов, диапазон достаточно велик — от $0 (бесплатная программа) до $999. Цена приложения обычно не достигает 5 долларов, а цена в 10 долларов считается уже очень большой.

Примечание

Раз уж речь зашла о прибыли, стоит повторить, что размер минимальной выплаты составляет 150 долларов, и если будет продано всего несколько экземпляров приложения, то создатель программы ничего не получит до тех пор, пока на его счету не наберется указанная сумма.

Популярные, по меркам App Store, цены приведены в табл. 11.1. Выбираем цену программы **Tier 1**, что по таблице соответствует 0,99 долларов.

Далее необходимо ввести информацию о программе, как показано на рис. 11.6.

В *главе 10* мы создали русскоязычное описание программы и отдали его на перевод, настало время использовать полученный английский текст. (Поскольку основной рынок сбыта — англоязычные страны, качественный перевод является одним из важных факторов успешных продаж программы, коряво переведенный текст может оттолкнуть часть потенциальных покупателей.)

Таблица 11.1. *Таблица цен Pricing Matrix*

Название	Цена, USD	Выплата, USD
Tier 1	0,99	0,70
Tier 2	1,99	1,40
Tier 3	2,99	2,10
Tier 4	3,99	2,80
Tier 5	4,99	3,50
Tier 6	5,99	4,20

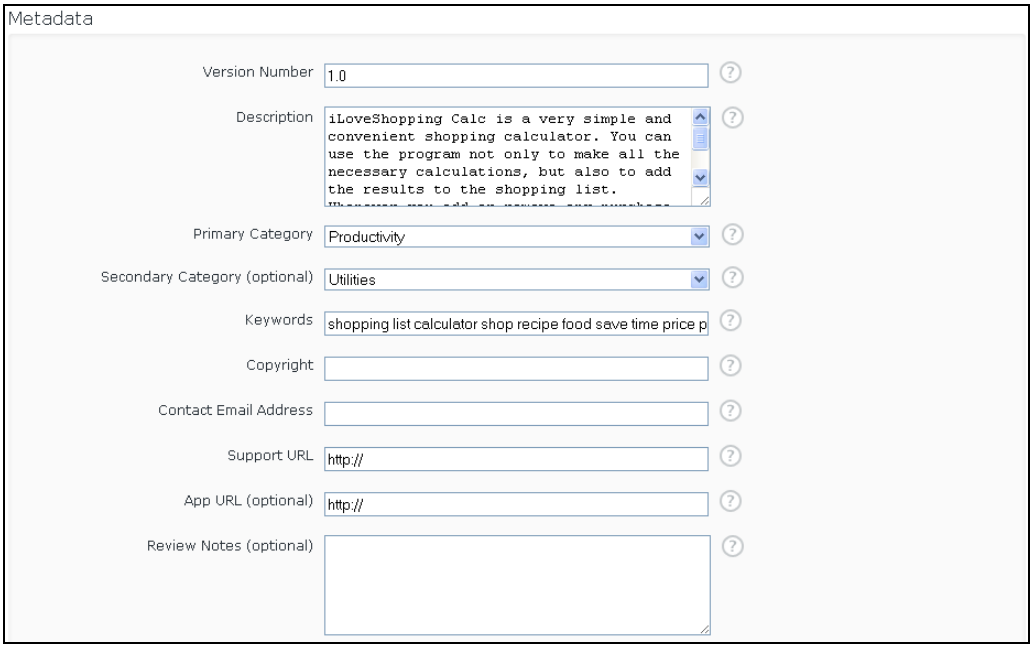


Рис. 11.6. Информация о программе

При вводе информации о программе следует учесть важные данные:

- **Version Number** — номер версии, желательно чтобы он совпадал со значением **Bundle Version**, указанным в plist-файле;
- **Primary Category** — основная категория, в которой будет размещена программа (приходится подбирать наиболее близкую по смыслу категорию);
- **Keywords** — ключевые слова, по которым пользователи смогут найти программу. Неправильно подобранные ключевые слова могут значительно уменьшить прибыль даже при качественном приложении.

Поля **Copyright**, **Contact Email Adress**, **Support URL**, **App URL** в комментариях не нуждаются. Опциональное поле **Review Notes** позволяет написать заметки

для того, кто будет тестировать программу, например, сделать подсказки по прохождению уровней игры или поместить другую полезную для тестирования информацию (в нашем приложении никакой дополнительной информации не требуется).

Возрастные ограничения и информация о методах шифрования программы в нашем случае отсутствуют. Программа создана, и запись появляется в окне **iTunes Connect.**

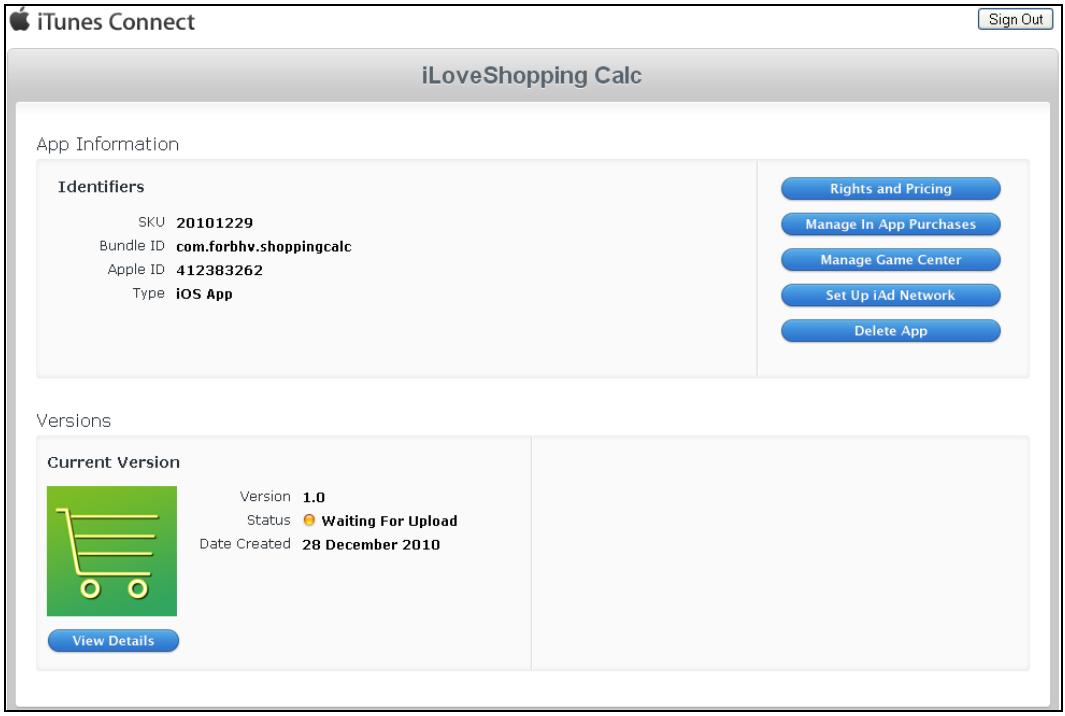


Рис. 11.7. Страница программы в окне **iTunes Connect**

Статус программы вначале показан как **Prepare for Upload**. Для того чтобы загрузить программу на сайт, нужно в разделе **View Details** щелкнуть кнопку **Ready to Upload Binary**. При этом будет задан вопрос, нет ли в программе шифрования, на который (в нашем случае) можно ответить отрицательно. После этого статус программы изменится на **Waiting For Upload**, как видно на рис. 11.7, и только теперь можно загрузить программу на сайт.

В более ранних версиях iTunes Connect загрузка происходила непосредственно на сайт, позже был разработан загрузчик Application Loader. Необходимо найти эту программу в папке установки XCode и запустить ее. Если до сих пор все сделано правильно и существует программа со статусом **Waiting for Upload**, то в окне **Application Loader** нам позволят выбрать эту программу, как показано на рис. 11.8. Если же имя программы в окне выбора не появляется, значит, какое-то из предыдущих действий не увенчалось успехом.

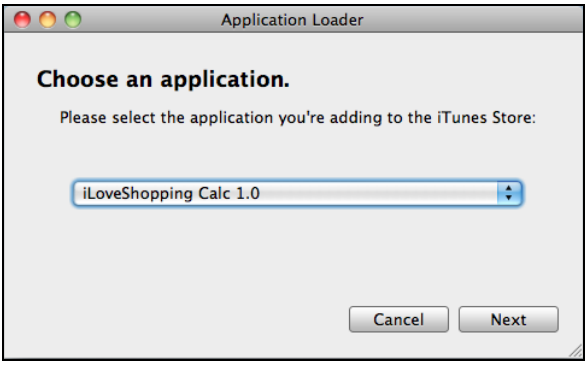


Рис. 11.8. Приложение Application Loader

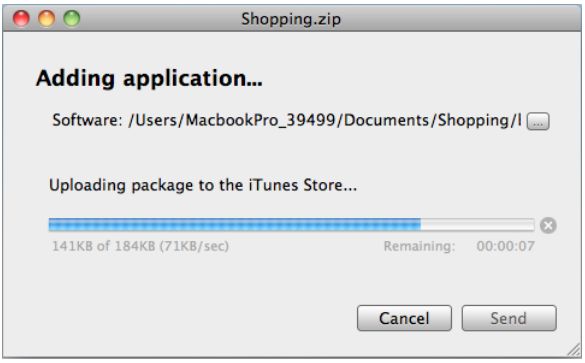


Рис. 11.9. Загрузка программы в App Store

Необходимо указать путь к zip-файлу, который мы запомнили *в главе 10*. Если все сделано правильно, начинается загрузка приложения, как показано на рис. 11.9.

Вот теперь размещение завершено, и с этого момента программа ставится в очередь на рассмотрение в App Store.

Результаты и статистика

Как и планировалось в самом начале, результаты размещения программы показаны "как есть", без каких-либо сокращений (табл. 11.2).

***Таблица 11.2.** Результаты рассмотрения программы*

Дата и время	Статус
28.12.2010 22:55	Waiting For Upload
29.12.2010 01:55	Upload Received
03.01.2011 08:47	In Review
03.01.2011 09:49	Rejected

Очередь на рассмотрение прошла довольно быстро, но программа не была допущена, пришло письмо от Apple следующего содержания:

```
Thank you for submitting iLoveShopping Calc to the App Store.


We've completed the review of your app, however, we cannot post this version
to the App Store because it is using a private API. Using private APIs can
lead to a poor user experience should these APIs change in the future and is
therefore not permitted, as noted in the App Store Review Guidelines
<https://developer.apple.com/appstore/resources/approval/guidelines.html>


2.5   Apps that use non-public APIs will be rejected


The non-public API that is included in your application is addTextFieldWithVa-
lue:label:. If you have defined a method in your source code with the same
name as this API, we suggest altering your method name so that it no longer
collides with Apple's private API to avoid your application being flagged in
future submissions.
```

Как понятно из текста, причиной отказа стало использование недокументированной функции `addTextFieldWithValue` (см. рис. 10.5). В этом есть здравый смысл: недокументированные функции могут быть изменены в новой версии ОС, что привело бы к неработоспособности приложений, которые эти функции используют. Следовательно, приложение нужно изменить.

Решение было найдено довольно быстро — создать класс, наследуемый от `UIAlertView`, который будет содержать дополнительное поле `UITextField` для ввода текста пользователем. Помимо этого потребовалась небольшая корректировка размера окна в классе `TextAlertView` файла проекта (в архиве примеров ShoppingCalc.zip). Важно то, что класс использует стандартные API-функции, разрешенные Apple.

Таким образом, замена класса `UIAlertView` на `TextAlertView` потребовала минимальной переделки кода программы. Продолжение изменения статуса показано в табл. 11.3.

Таблица 11.3. *Ход рассмотрения программы*

Дата	**Статус**
04.01.2011 09:53	Waiting For Upload
04.01.2011 09:57	Upload Received
06.01.2011 07:31	In Review
06.01.2011 14:48	Processing for App Store

Надо заметить, что рассмотрение программы прошло удивительно быстро. И вот теперь можно перейти к самому интересному — к данным о продажах.

Результаты продаж

Автор может раскрыть большую коммерческую тайну и привести результаты продаж программы полностью в табл. 11.4.

Таблица 11.4. *Результаты продаж приложения*

Дата	Количество экземпляров
07.01.2011	2
08.01.2011	2
09.01.2011	2
10.01.2011	4
11.01.2011	0
12.01.2011	1
13.01.2011	1
14.01.2011	2
15.01.2011	0
16.01.2011	0
17–23 января 2011 г.	0
24–30 января 2011 г.	0
31 января–6 февраля 2011 г.	0
7–13 февраля 2011 г.	5
14–20 февраля 2011 г.	8
21–27 февраля 2011 г.	0
28 февраля–06 марта 2011 г.	4

Таким образом, за два месяца было продано 26 экземпляров программы, что принесло прибыль в 18 долларов: мало для коммерческого проекта, но вполне достаточно для проекта учебного. Более того, результаты продаж программы имеют важное значение: они показывают, что в App Store даже простая программа все равно имеет шансы на продажу.

Приведем для сравнения график загрузок другой программы, распространяемой автором бесплатно (рис. 11.10). Важный и очевидный момент — загрузки максимальны в начале, когда программа находится в первых страницах на сайте и в поиске. Затем, по мере появления новых программ, количество загрузок уменьшается, со временем доходя до нуля.

К сожалению, приходится признать, что программный продукт в App Store либо "выстрелит" и принесет основную прибыль в первый месяц, либо "пройдет мимо" и останется фактически незамеченным. Хотя, конечно, никто не запрещает исполь-

зовать различные вспомогательные средства — рекламу в Google, обзоры на специализированных сайтах и пр.

Важное примечание

Выпуск новой версии не переводит программу в верхнюю часть списка результатов поиска по ключевым словам. Сначала это кажется странным, но ведь в противном случае началась бы настоящая "гонка" среди разработчиков программ по выпуску незначительных обновлений. Для пользователей это было бы крайне неудобно.

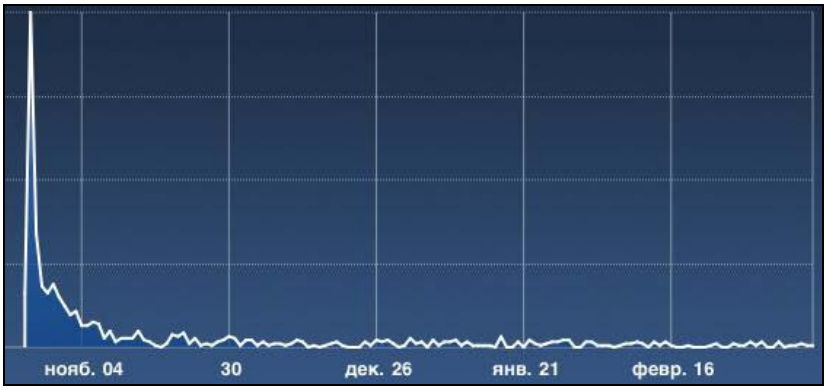


Рис. 11.10. График загрузки бесплатного приложения из App Store

Не менее интересен вопрос о рынках сбыта программы. Количество продаж нашего приложения слишком мало для получения точной статистики, поэтому в табл. 11.5 приведены обобщенные данные по нескольким программным продуктам.

Таблица 11.5. *Статистика по странам*

Страна	**Процентное соотношение**
США	40,6 %
Великобритания	9,8 %
Австралия	4,9 %
Канада	4,8 %
Германия	4,0 %
Италия	3,1 %
Япония	2,7 %
Франция	2,6 %
Мексика	2,3 %
Россия	2,0 %
Прочие	22 %

Мы заканчиваем главу, посвященную продажам. В заключение хочется ответить на вопрос, стоит ли браться за самостоятельную продажу программ в App Store. Ответ таков: стоит, но лишь в том случае, если разработчик чувствует в себе уверенность и финансовую возможность выпустить программу достойного уровня. Программа должна быть действительно хорошей, планка качества сейчас высока. Кроме того, любой новый и интересный проект — это дополнительные опыт и знания, которые всегда пригодятся и могут принести косвенную прибыль, например в резюме при устройстве на работу. Удовольствие и полученный опыт, по мнению автора, даже ценнее заработанных денег.

Полагая, что читатели имеют достаточно информации, чтобы дальше двигаться самостоятельно, автор желает в этом процессе творческих успехов и счастливого пути…

Заключение

Мы заканчиваем рассмотрение вопросов, связанных с программированием для операционной системы iOS, и адресуем читателя к разделу **iOS Dev Center** сайта **http://developer.apple.com**.

Конечно, нельзя объять необъятное, объем книги ограничен. Остается дать ссылки на те разделы, которые весьма важны и актуальны для изучения.

- Поддержка экранов высокого разрешения Retina с разрешением 640×960 описана в разделе **Specifying High-Resolution Images in iOS** на сайте Apple.
- Поддержка покупок "из программы" с помощью технологии In-App Purchase подробно описана на русском языке по адресу **http://habrahabr.ru/blogs/macosxdev/84359/**.
- Поддержка технологии GameKit дана в разделе **Game Kit Programming Guide** сайта Apple. Там же можно скачать готовые примеры GKTank и GKRocket, доступные в исходных кодах.
- Желающие изучить технологию трехмерной графики OpenGL могут обратиться к русскоязычному описанию в виде 14 уроков на сайте **http://lookapp.ru**. Первый урок находится по адресу **http://lookapp.ru/2009/06/15/iphone-sdk-tutorials59/**.
- Ресурс **LookApp.ru** содержит переводы уроков по iOS SDK и начинается со статьи "Устанавливаем XCode" по адресу **http://lookapp.ru/2009/06/04/iphone-sdk-tutorials46/**. Желающие получить ответы на вопросы по устройствам Apple и по программированию для них также могут обратиться к русскоязычному форуму по адресу **http://www.iphones.ru/forum/**.

В завершение хочется выразить благодарность родным и близким, терпящим многочасовое времяпровождение автора за компьютером. Без их поддержки эта книга вряд ли вышла бы в свет.

Приложение

Список примеров, использованных в книге

Имя файла	Описание
Glava02_s1HelloWorld.zip	Глава 2. Проект "Hello World"
Glava03_s1UIButton.zip	Глава 3. Использование класса `UIButton`
Glava03_s2UIButtonMore.zip	Глава 3. Использование класса `UIButton`, дополнительные возможности
Glava03_s3UIScroller.zip	Глава 3. Использование класса `UIScroller`
Glava03_s4UIInput.zip	Глава 3. Использование класса `UITextField`
Glava03_s5UIInputMore.zip	Глава 3. Расширенные возможности ввода с использованием `UITextField`
Glava03_s6UITable.zip	Глава 3. Использование класса `UITableView`
Glava03_s7UIAlert.zip	Глава 3. Использование класса `UIAlertView`
Glava03_s8UIImage.zip	Глава 3. Использование класса `UIImage`
Glava04_s1UIViews.zip	Глава 4. Переключение окон в программе
Glava04_s2UIViewsDelegate.zip	Глава 4. Переключения окон в программе с посылкой уведомлений
Glava04_s3UITableCustom.zip	Глава 4. Создание таблиц с нестандартными ячейками
Glava04_s4UITableSegments.zip	Глава 4. Создание таблиц с сегментами
Glava04_s5UIWebView.zip	Глава 4. Использование класса `UIWebView`
Glava05_s1UITouch.zip	Глава 5. Обработка касаний сенсорного экрана
Glava05_s2TestDraw.zip	Глава 5. Вывод с помощью графических примитивов
Glava06_s1UITouchMulti.zip	Глава 6. Обработка касаний с помощью технологии Multi Touch
Glava06_s2TestAccel.zip	Глава 6. Акселерометр
Glava06_s3TestInternet.zip	Глава 6. Загрузка данных из сети Интернет
Glava06_s4ScrRotation.zip	Глава 6. Поворот экрана

(окончание)

Имя файла	**Описание**
Glava06_s5SwitchFullScreen.zip	Глава 6. Переключение в полноэкранный режим
Glava06_s6PlaySounds.zip	Глава 6. Воспроизведение звука
Glava07_s1PhotoAlbum.zip	Глава 7. Загрузка данных из фотоальбома
Glava07_s2Contacts.zip	Глава 7. Список контактов
Glava07_s3TestBattery.zip	Глава 7. Получение состояния батареи
Glava07_s4TestFiles.zip	Глава 7. Функции работы с файлами
Glava08_s1UIGame.zip	Глава 8. Игра с использованием элементов интерфейса операционной системы
Glava08_s2Box2DHelloWorld.zip	Глава 8 Программа Hello World с использованием библиотеки Box2D
Glava08_s3Box2DGame.zip	Глава 8. Игра, созданная с использованием библиотеки Box2D
Glava08_s4Cocos2DHelloWorld.zip	Глава 8. Программа Hello World, созданная с помощью библиотеки Cocos2D
Glava08_s5Cocos2DGameSrc.zip	Глава 8. Игра, созданная с помощью библиотеки Cocos2D
Glava09_ShoppingCalc.zip	Глава 9. Программа Shopping Calc

Предметный указатель