Дэн ФЕРНАНДЕС
Брайан ПИК

программируем для удовольствия

10 .NET-проектов для Wiimote, World of Warcraft, YouTube

Символ®

Coding4Fun

10 .NET Programming Projects for Wiimote, World of Warcraft, YouTube, and More

Dan Fernandez, Brian Peek

O'REILLY®

Coding4Fun

программируем для удовольствия

10 .NET-проектов для Wiimote, World of Warcraft, YouTube и других программ

Дэн Фернандес, Брайан Пик

Символ®

Санкт-Петербург — Москва
2009

Дэн Фернандес, Брайан Пик

Coding4Fun: программируем для удовольствия. 10 .NET-проектов для Wiimote, World of Warcraft, YouTube и других программ

Перевод А. Слинкина

Главный редактор	*А. Галунов*
Зав. редакцией	*Н. Макарова*
Выпускающий редактор	*П. Щеголев*
Редактор	*Ю. Бочина*
Корректор	*О. Макарова*
Верстка	*Д. Орлова*

Фернандес Д., Пик Б.

Coding4Fun: программируем для удовольствия. 10 .NET-проектов для Wiimote, World of Warcraft, YouTube и других программ. – Пер. с англ. – СПб.: Символ-Плюс, 2009. – 480 с., ил.

ISBN 978-5-93286-166-0

Издание предназначено для тех разработчиков, кто хочет отвлечься от решения коммерческих бизнес-задач и сделать несколько проектов для души. Для тех, кто хочет вспомнить о том, с чего начиналось их увлечение программированием, кому хочется заинтересовать им своего ребенка или просто развлечься, написав несколько небольших забавных проектов.

Книга охватывает разные сферы, так что всякий найдет себе хотя бы один проект по душе. Вы бы хотели самостоятельно написать игру для Xbox, создать электронную доску с помощью пульта Wiimote или разработать собственное пиринговое приложение? Эта книга поможет приступить к работе с некоторыми из самых современных программных и аппаратных платформ, применяя бесплатное ПО от корпорации Майкрософт. С нею можно программировать на языках C# и VB в таких средах, как Lua, ASP.NET, WPF, XNA Game Studio и Popfly. Главы совершенно не зависят друг от друга – в каждой рассматривается новая тема и объясняется, как читатель может сделать это самостоятельно и какие навыки, умения или оборудование могут ему потребоваться. В книге много схем и иллюстраций, примеров кода и пошаговых инструкций. Она научит программировать по-новому, к тому же весело и с удовольствием.

ISBN 978-5-93286-166-0
ISBN 978-0-596-52074-8 (англ)

Издательство «Символ-Плюс». 199034, Санкт-Петербург, 16 линия, 7,
тел. (812) 324-5353, www.symbol.ru. Лицензия ЛП N 000054 от 25.12.98.

Подписано в печать 20.08.2009. Формат 70×100$^{1}/_{16}$. Печать офсетная.
Объем 30 печ. л. Тираж 1500 экз. Заказ №
Отпечатано с готовых диапозитивов в ГУП «Типография «Наука»
199034, Санкт-Петербург, 9 линия, 12.

Маме, которая всегда поддерживает и подбадривает меня;
папе, который приложил все силы к тому, чтобы я стал таким,
какой есть, и Энджи – любви всей моей жизни
и женщине моей мечты.

Дэн Фернандес

В память о любимом отце. Ты слишком рано покинул нас,
нам всем так тебя недостает. Маме, опоре моей жизни,
которая остается сильной и не сдается. И Мишель, моей любви
и лучшему другу, которая совершила все возможное и даже больше,
чтобы мы смогли это пережить, и благодаря которой
каждый новый день становится лучше предыдущего.

Брайан Пик

Оглавление

Предисловие

Разработка программного обеспечения может принимать разные формы. Для многих профессиональных программистов она зачастую означает монотонную работу по реализации приложений, которые опрашивают базу данных и генерируют длинные отчеты. Для студентов и любителей создание программ может принимать форму несложных упражнений на сортировку списков или рисование картинок на экране.

Но вне зависимости от вашей квалификации и профессии построение компьютерных программ может быть гораздо более увлекательным и одновременно познавательным делом. Почему бы вместо перетаскивания элементов управления на форму и выборки заказов из базы не написать видеоигру или не погонять дистанционно управляемую машинку с помощью пульта Wii Remote от приставки Nintendo?

Идея этой книги в том и заключается, чтобы показать, как можно создать программы, которые делают забавные и интересные вещи. Мы рассмотрим 10 разных проектов; после их проработки у вас будет, с чем поиграть и чем похвастаться перед друзьями и родственниками. Здесь вы найдете учебные пособия, диаграммы, фрагменты кода, фотографии и вообще все необходимое для того, чтобы превратить разработку ПО в удовольствие, а заодно кое-чему научиться.

Хотите подключить к компьютеру внешнее оборудование и нестандартные устройства ввода? Мы расскажем, как это сделать. Хотите написать видеоигру, которая будет работать на вашей машине, а кроме того на приставке Xbox 360 и на карманном медиаплеере Zune? Поможем и в этом. Желаете поработать с популярными социальными сетями типа YouTube и Twitter новыми интересными способами? Читайте дальше. Вам нужно вытащить из базы последние 10 заказов клиента? Извини, друг, ты ошибся адресом.

На кого рассчитана эта книга?

Эта книга ориентирована на самых разных пользователей и разработчиков ПО. Если вы знакомы с разработкой в среде .NET на языке C# или Visual Basic, то сможете одолеть все приведенные в ней проекты. Если вы просто пользуетесь компьютером и с программированием не очень знакомы, то все равно сможете довести до конца несколько про-

ектов без особых затруднений. Вне зависимости от опыта и знаний, если вы захотите сразу перейти к готовому продукту, минуя все промежуточные стадии, то для каждой главы сможете скачать сопровождающее ее готовое к работе приложение.

Поэтому не пугайтесь – быть кандидатом физико-математических наук совершенно необязательно. Для читателя любого уровня подготовки в этой книге найдется проект, который он сможет выполнить полностью, и еще несколько, позволяющих расширить знания и навыки, – и при этом получить удовольствие!

Что необходимо для чтения этой книги

Для чтения разных глав понадобятся разные знания и устройства, но некоторые программы необходимо скачать и установить с самого начала.

Во-первых, следует установить какую-нибудь редакцию Visual Studio 2008. Если вы профессиональный разработчик, то, возможно, Visual Studio на вашем компьютере уже стоит. Если нет, тоже не страшно, – редакция Visual Studio 2008 Express бесплатна, установить ее очень просто. Заметим, кстати, что Visual Studio 2008 Express без всяких конфликтов устанавливается на одной машине с Visual Studio 2005 (и Visual Studio 2005 Express).

Все редакции Visual Studio 2008 Express можно найти на сайте *http://www.microsoft.com/Express/download*.

Редакция Visual Studio 2008 Express поставляется в нескольких вариантах в зависимости от языка программирования и типа разрабатываемых приложений. Если вы собираетесь компилировать версии программ, написанных на C#, то должны установить Visual C# 2008 Express Edition. Если вы лучше знаете Visual Basic, то установите Visual Basic 2008 Express Edition. Для тех проектов из этой книги, где требуется веб-разработка, понадобится Visual Web Developer 2008 Express Edition.

В главах, где речь идет об аппаратуре, предполагается также, что у вас есть стандартные инструменты: отвертка, молоток, плоскогубцы, паяльник и т. д.

В некоторых главах понадобятся специальные компоненты и программы. Прочитайте следующий раздел, где написано, что конкретно требуется для завершения каждого проекта; тогда, приступая к проекту, вы хотя бы будете представлять, что вас ожидает. Кроме того, в каждой главе требования объясняются более подробно.

Все проекты были написаны и протестированы на ПК под управлением Windows XP SP2 или старше и Windows Vista. Если вы пользуетесь Mac или какой-то другой операционной системой, то, возможно, некоторые проекты должны будут выполняться на виртуальной машине

(например, в среде VMWare или Parallels); при этом мы не можем гарантировать успешности процесса и в случае чего помочь не сможем.

Как организована эта книга

Часть I. Игры

Глава 1. Атака из космоса

В этой главе рассказывается, как написать простую двумерную видеоигру с помощью программы XNA Game Studio. Игра будет работать на любой Windows-машине с DirectX9 при наличии видеокарты, поддерживающей спецификацию Pixel Shader 1.1 или выше, а также на видеоприставке Xbox 360 и на медиаплеере Microsoft Zune. Для этого проекта понадобится скачать и установить бесплатную программу XNA Game Studio 3.0 производства Microsoft. Понятно, что для запуска игры на Xbox 360 или Zune нужно иметь соответствующее устройство, но если его нет, можно собрать версию для Windows и запускать ее без всякого дополнительного оборудования или ПО.

Глава 2. Солдат LEGO: двумерная игра с боковым скольжением на платформе Popfly

В этой главе показано, как с помощью Popfly – бесплатного комплекта основанных на технологии Silverlight инструментов для создания веб-сайтов, Mashup-приложений и игр – написать двумерную игру для среды Silverlight. Необходимо установить среду исполнения Silverlight. Для создания собственных фигурок LEGO потребуется скачать и установить программу Digital Designer компании LEGO.

Глава 3. Чтение новостей: встраиваемый модуль чтения RSS-лент для World of Warcraft

В этой главе рассматривается построение Feed Reader – встраиваемого в игру *World of Warcraft* модуля чтения новостей из RSS-лент. Для программирования этой надстройки рекомендуется установить программу Add-On Studio для World of Warcraft версии 2.0. Чтобы воспользоваться модулем Feed Reader, понадобится установить *World of Warcraft*.

Часть II. Программы для веб

Глава 4. InnerTube: скачиваем, конвертируем и синхронизируем видеоролики с YouTube

В этой главе мы займемся приложением InnerTube, которое умеет скачивать, конвертировать и синхронизировать (записывать на мобильное устройство) видеоролики с YouTube. InnerTube может синхронизироваться со следующими устройствами: iPod, iPhone, Zune.

Глава 5. PeerCast: смотрим видео со своего компьютера из любой точки планеты

В этой главе вы узнаете о приложении PeerCast, которое позволяет транслировать потоковое видео с вашего домашнего ПК на другой ПК в любой точке света. Поскольку PeerCast обменивается данными по протоколу Peer Channel, то ваш маршрутизатор должен этот протокол поддерживать.

Глава 6. TwitterVote

В этой главе мы разработаем веб-приложение, которое позволит создавать онлайновые опросы в социальной сети Twitter, а затем просматривать результаты в виде гистограммы. Кроме Visual Web Developer 2008 Express и среды исполнения Silverlight, никаких других программ или аппаратуры не потребуется.

Глава 7. WHSMail: надстройка над Windows Home Server для чтения почты в Outlook

В этой главе демонстрируется создание надстройки над Windows Home Server, которая позволяет в любой момент просматривать через Интернет хранящуюся в Outlook почту. Для этого проекта необходима операционная система Windows Home Server (запустить программу можно в любой системе, включающей Internet Information Services, но функциональность будет ограничена).

Часть III. Технические устройства

Глава 8. Машинка, управляемая с пульта Wiimote

В этой главе шаг за шагом описывается, как можно применить пульт Wii Remote (Wiimote) для управления игрушечной машинкой. Вам потребуется:

- ПК с поддержкой Bluetooth
- Пульт Nintendo Wii Remote (Wiimote)
- Дистанционно управляемая машинка с цифровыми входами
- Паяльник и припой
- Провода

Глава 9. Электронная доска и Wiimote

В этой главе вы создадите интерактивную электронную доску, управляемую с пульта Wii Remote (Wiimote). Вам потребуется:

- Пульт Nintendo Wii Remote (Wiimote)
- Инфракрасный светодиод
- Нормально разомкнутый быстродействующий переключатель
- Батарейка AA или AAA
- Маркер-ластик
- Паяльник и припой
- Дремель

Глава 10. Анимированное музыкальное светошоу

В этой главе вы найдете инструкции по сборке установки для демонстрации светового шоу в такт музыке, как на знаменитых праздничных светошоу, которые вы могли видеть на YouTube. Вам потребуется:

- Библиотека Phidget API (скачивается бесплатно)
- Пять шнуров-удлинителей с двухштырьковыми вилками для использования вне помещения на каждую интерфейсную плату
- Стандартный провод калибра 14–16
- Две муфты под провод калибра 14–16 на каждую интерфейсную плату
- Корпус, способный вместить собранные интерфейсные платы
- Новогодние гирлянды лампочек
- Внешние динамики или FM-передатчик, чтобы можно было слушать музыку.

Типографские соглашения

В книге применяются следующие выделения:

Курсив

Таким начертанием выделяются новые термины, URL, команды, имена и расширения имен файлов, имена папок и UNC-имена.

`Моноширинный шрифт`

Применяется для выделения команд, опций, флагов, переменных, атрибутов, ключей, функций, типов, классов, пространств имен, методов, модулей, свойств, параметров, значений, объектов, событий и их обработчиков, XML- и HTML-тегов, макросов, содержимого файлов и результатов выполнения команд.

`Моноширинный курсив`

Текст, вместо которого надо подставить значения, вводимые пользователем.

`Моноширинный полужирный шрифт`

Команды или другой текст, который пользователь должен вводить буквально.

Примечание

Совет, рекомендация или общее замечание.

Внимание

Предупреждение или предостережение.

О примерах кода

Для тех проектов, где требуется программировать, имеются примеры как на Visual Basic, так и на C#, за исключением главы об XNA, для которой необходим C#. Весь код написан в Visual Studio 2008 SP1 с .NET 3.5 SP1.

И код, и решение для Visual Studio к каждой главе можно найти на сайте этой книги по адресу *http://www.c4fbook.com/*. Многие проекты будут обновляться и после выхода книги в свет, поэтому заглядывайте на сайт – там могут появиться новые, улучшенные версии приложений с исправленными ошибками и дополнительными функциями.

Использование примеров

Эта книга призвана помогать вам в работе. Поэтому вы можете использовать приведенный в ней код в собственных программах и в документации. Спрашивать у нас разрешение необязательно, если только вы не собираетесь воспроизводить значительную часть кода. Например, не требуется разрешение, чтобы включить в свою программу несколько фрагментов кода из книги. Однако для продажи или распространения примеров на компакт-диске нужно получить разрешение. Можно без ограничений цитировать книгу и примеры в ответах на вопросы. Но чтобы включить значительные объемы кода в документацию по собственному продукту, нужно получить разрешение.

Мы высоко ценим, хотя и не требуем, ссылки на наши издания. В ссылке обычно указываются название книги, имя автора, издательство и ISBN, например: «Coding4Fun by Dan Fernandez and Brian Peek. Copyright 2009 Dan Fernandez and Brian Peek, 978-0-596-52074-8.».

Если вы полагаете, что планируемое использование кода выходит за рамки изложенных выше границ использования, пожалуйста, обратитесь к нам по адресу *permissions@oreilly.com*.

Как с нами связаться

Мы тщательно проверили всю изложенную в этой книге информацию, но вполне возможно, что некоторые функции программ изменились или мы допустили ошибку (ужасно и невозможно поверить). Пожалуйста, сообщайте нам обо всех обнаруженных ошибках и присылайте свои предложения по поводу будущих изданий на адрес:

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

Для этой книги есть веб-страница, на которой выкладываются примеры, рисунки и планы относительно последующих изданий. Адрес страницы:

http://www.c4fbook.com/

или

http://www.oreilly.com/catalog/9780596520748

Можно также отправлять сообщения в электронном виде. Если вы хотите, чтобы вас включили в список рассылки или выслали вам бесплатный каталог, отправьте сообщение на адрес:

authors@c4fbook.com

или

info@oreilly.com

Касающиеся книги замечания следует отправлять по адресу:

authors@c4fbook.com

или

bookquestions@oreilly.com

Дополнительную информацию о наших книгах, конференциях, ресурсных центрах и сети O'Reilly Network можно найти на сайте:

http://www.oreilly.com

Safari® Books Online

Если на обложке технической книги есть пиктограмма «Safari® Books Online», это означает, что книга доступна в Сети через O'Reilly Network Safari Bookshelf.

Safari предлагает намного лучшее решение, чем электронные книги. Это виртуальная библиотека, позволяющая без труда находить тысячи лучших технических книг, вырезать и вставлять примеры кода, загружать главы и находить быстрые ответы, когда требуется наиболее верная и свежая информация. Она свободно доступна по адресу *http://safari.oreilly.com*.

Благодарности

Как всегда, для того чтобы наши идеи предстали в виде переплетенной стопки бумаги, которую вы держите в руках (или читаете на экране монитора), потрудиться пришлось множеству людей.

Во-первых, огромная благодарность нашему редактору, прелестной и талантливой Лорель Рума (Laurel Ruma), которая мирилась с нашей тупостью и подгоняла нас, когда мы начинали лениться. Работа с тобой – просто фантастика!

Мы благодарны также авторам, нашедшим время написать о собственных проектах, которые помогли улучшить нашу изначальную идею.

Наши технические рецензенты также заслуживают всяческих похвал. На вычитку текста и тщательное тестирование программ уходило очень много времени. Во многих случаях приходилось мастерить устройства, тестировать версии кода на двух языках и повторять тестирование по мере того, как мы вносили усовершенствования. Спасибо вам, Крис Чарабарук (Chris Charabaruk), Дженни Чоудберри (Jenny Chowdhury), Энди Данн (Andy Dunn), Майкл Итон (Micheal Eaton), Джейк Гуд (Jake Good), Джонни Хэлайф (Johnny Halife), Эд Кейм (Ed Kaim), Брайан Келлер (Brian Keller), Майк Корчински (Mike Korcynksi), Майкл Леттерль (Michael Letterle), Джио Монтроне (Gio Montrone), Адам Натан (Adam Nathan), Скотт К. Рейнольдс (Scott C. Reynolds), Майкл Сэмпсон (Michael Sampson), Гриффит Таунсенд (Griffith Townsend), Дин Вэбер (Dean Weber), Крис Дж. Уилльямс (Chris G. Williams) и Лу Вега (Lou Vega), – вы приложили максимум усилий к тому, чтобы эта книга стала лучше.

От Дэна Фернандеса

Благодарю Брайана Пика и Лорель Рума за то, что они терпели меня. Мы хохотали, мы спорили, но все-таки довели это дело до конца!

Отдельное спасибо моей жене Энджи, которая тащила мое истекающее кровью, полумертвое тело к финишной черте, знаменующей окончание работы над книгой. Я не смог бы написать ее без твоей любви, заботы и участия.

Я благодарен многим людям, которые способствовали появлению этой книги на свет, в том числе (имена перечисляются без какого-то определенного порядка): Адаму Кинни (Adam Kinney) за его знание WPF, Дэниэлу Моту (Daniel Moth), который помог мне разобраться с чертовыми пространствами имен в VB, Робу Шлендеру (Rob Schlender) за помощь в освоении библиотеки C4F P2P, Брайану Келлеру (Brian Keller) за техническое рецензирование и помощь в создании веб-сайта Coding4Fun в 2004 году, Клинту Руткасу (Clint Rutkas) за планомерное сопровождение этого сайта, Габору Ратки (Gabor Ratky) за AddOn Studio и глубокое знание игры *Warcraft*, Джеффу Сэндквисту (Jeff Sandquist) за терпение и мудрость. Также благодарю сайт BassDrive.com за бодрую музыку, не дававшую мне уснуть в предрассветные часы, участников проекта The Uldum Cabal, прощавших мое отсутствие в *Warcraft*, Бамбино, который составлял мне компанию, и, конечно, всех верных читателей и авторов сайта Coding4Fun.

От Брайана Пика

Первым делом благодарю Дэна Фернандеса за то, что с ним было так здорово работать, и за юмористическое отношение к трудностям. Жду не дождусь следующего совместного проекта...

Спасибо моей любимой подруге Мишель Ливитт (Michelle Leavitt) – самой заботливой, любящей женщине в моей жизни, моей поддержке и опоре. Неважно, хорошо или плохо идут дела, – ты всегда оказываешься рядом, чтобы все поправить. Твоя безграничная любовь и непрестанная поддержка делают меня лучше. Я люблю тебя, и никакие написанные в этом абзаце слова не могут рассказать, насколько сильно!

Спасибо маме за то, что она поддерживает меня во всех начинаниях и особенно в этом. Не унывай. Все образуется...

Большое спасибо моим друзьям, Джоуи Бучеку (Joey Buczek), который сделал все рисунки, и Дэвиду Уоллиману (David Wallimann), написавшему музыку и создавшему звуковые эффекты для главы «Атака из космоса». Не забудьте заглянуть на их сайты *http://www.joeybphotography.com/* и *http://www.davidwallimann.com/*, там вы найдете немало интересных фотографий, картинок и музыки.

Благодарю Лесли Сэнфорд (Leslie Sanford) (*http://www.lesliesanford.com/*) за предоставленный инструментарий MIDI, которым мы пользовались в главе «Анимированное музыкальное светошоу».

И, наконец, огромное спасибо всем моим друзьям и родственникам, которые поддерживают меня в жизни: Майку Килкуллену (Mike Kilcullen), Мэтту Кеннеди (Matt Kennedy), Марку Зоггу (Mark Zaugg) (такому сладкоречивому и не утратившему канадского своеобразия), Джонатану Гудьиру (Jonathan Goodyear), Джио Монтроне (Gio Montrone), Дэйву Бэнкрофту (Dave Bancroft), Патти Росси (Patti Rossi), Ардену Раучу (Arden Rauch), Роберту Шарле (Robert Sharlet), Меган Поттер (Megan Potter), м-ру Биверу (Mr. Beaver) и м-ру Тэртли (Mr. Turtley).

I

Игры

1

Атака из космоса

Автор	Брайан Пик
Сложность	Средняя
Необходимое время	5 часов
Стоимость	для Windows – бесплатно; для Xbox 360 – стоимость оборудования плюс $99 в год за членство в клубе создателей игр для XNA (но есть и способы попасть туда без денег, например программа бесплатного доступа DreamSpark для разработчиков-студентов); для Zune – стоимость устройства
Программное обеспечение	Visual C# 2008 Express или любая полная редакция Visual Studio 2008, а также XNA Game Studio 3.0
Оборудование	Версия для Xbox 360 требует наличия приставки Xbox 360 с жестким диском; версия для Zune – наличия медиаплеера Zune
Адрес в Интернете	*http://www.c4fbook.com/AlienAttack*

С выходом XNA Game Studio корпорация Microsoft подняла любительскую разработку игр на новый уровень. XNA – это простая в использовании интегрированная среда для разработки игр, позволяющая распространять их и запускать не только на ПК, но также на видеоприставке Xbox 360 и медиаплеере Zune. Кроме того, наличие единой среды и общего языка разработки делает XNA почти идеальным инструментом для воплощения идеи «пишем один раз, запускаем всюду». Код, написанный для Windows, практически всегда можно откомпилировать и запустить на Xbox 360 и Zune, не внося никаких или почти никаких изменений.

Краткий обзор

В этой главе мы воспользуемся средой XNA Framework для создания очень простой двумерной игры, которая будет работать на ПК, Xbox 360 и Zune. На примере написания клона классической аркадной игры *Space Invaders*, которую мы назвали «Атака из космоса» (Alien Attack) (рис. 1.1), будут рассмотрены различные вопросы, относящиеся к двумерной машинной графике, компьютерному воспроизведению звука, пользовательскому вводу и основам искусственного интеллекта (ИИ).

Имейте в виду, что игра будет работать на всех трех поддерживаемых платформах – Windows, Xbox 360, and Zune – без изменения кода при условии использования надлежащих директив компилятора. Эти директивы служат для исключения некоторых участков программы в зависимости от того, какая версия компилируется. По умолчанию проекты для Windows помечены строкой *WINDOWS*, проекты для Xbox 360 – *XBOX*, а проекты для Zune – *ZUNE*. Директива `#if` позволяет выделить те участки кода, которые относятся к конкретной платформе.

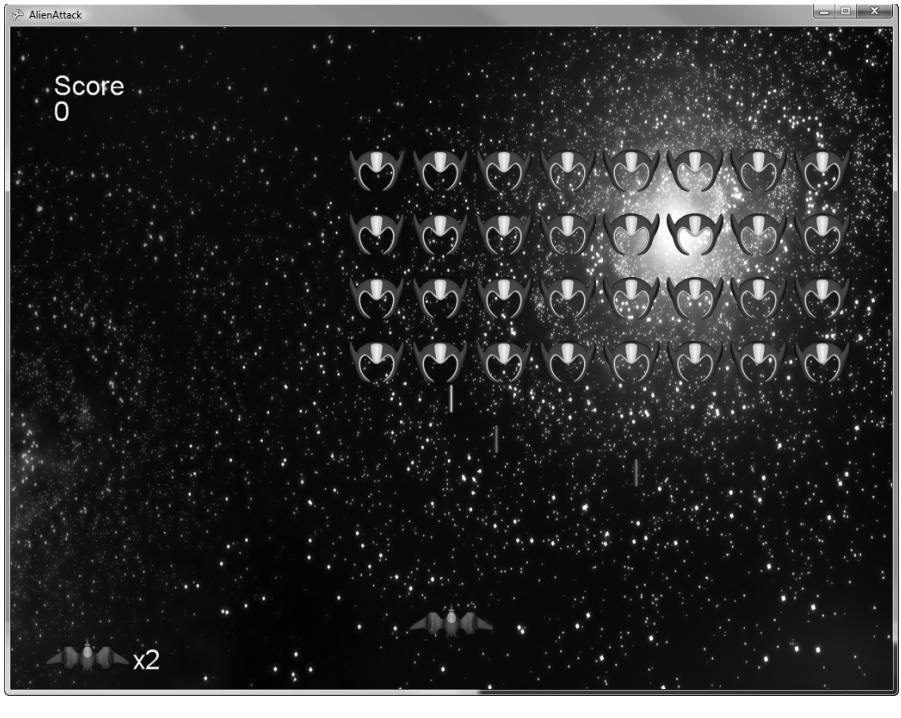

***Рис. 1.1.** Атака из космоса*

Подготовка

Проверьте, что установлена программа Visual C# 2008 Express или Visual Studio 2008. После этого скачайте последнюю версию XNA Game Studio 3.0 с сайта *http://creators.xna.com/*. Установите этот пакет и на вопрос, хотите ли открыть на брандмауэре порты, необходимые для развертывания на приставке Xbox 360, ответьте утвердительно, если планируете вести разработку для этой платформы. В результате на вашей машине будет установлено все необходимое для работы XNA и создано несколько шаблонов по умолчанию для разработки XNA-игр в Visual Studio.

Как обычно, полный исходный текст проекта можно скачать с сайта книги по адресу *http://www.c4fbook.com/AlienAttack*. Но даже если вы хотите написать игру с нуля, выполнив все описанные ниже шаги, все равно потребуется пакет «активов» (графика, звуковое сопровождение и т. д.). Он тоже есть на сайте. Скачайте архив и распакуйте его, чтобы активы были под рукой, когда они понадобятся.

Версия для Windows

Начнем с создания версии игры для Windows. Написание всех рассматриваемых XNA-игр начинается с разработки Windows-версии, которая затем переносится на Xbox 360 или Zune. Это упрощает отладку и развертывание.

Запустите Visual C# 2008 Express или Visual Studio 2008 и создайте новый проект типа *Windows Game (3.0)*, назвав его `AlienAttack`. Проект выбирается из группы `Visual C#→XNA Game Studio 3.0`, как показано на рис. 1.2.

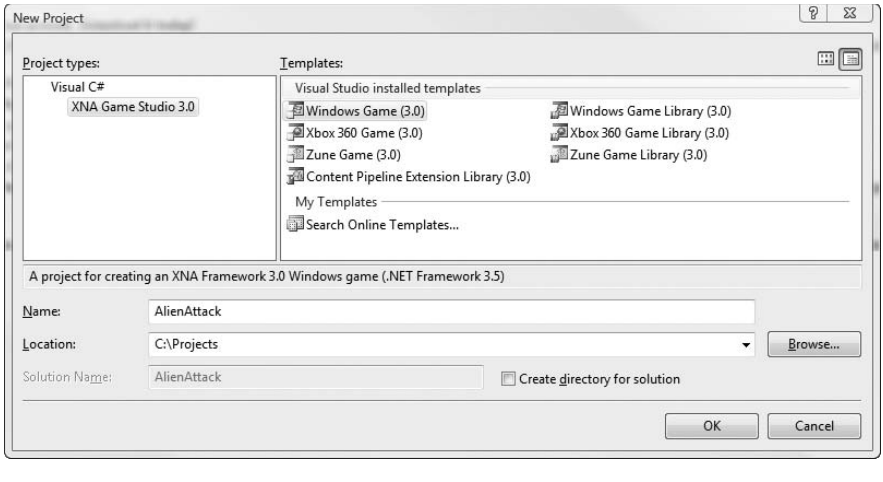

Рис. 1.2. *Создание проекта типа Windows Game (3.0)*

Будет создана заготовка проекта по умолчанию Windows Game. Запустите приложение, нажав клавишу F5 или выбрав из меню пункт Debug→ Start Debugging. Игра запустится, и вы увидите окно василькового цвета. Вам остается только добавить, гм... ну в общем все.

Для начала переименуйте файл *Game1.cs* в *AlienAttackGame.cs*. Интегрированная среда разработки (IDE) спросит, хотите ли вы изменить все встречающиеся в проекте ссылки на этот файл. Ответьте Yes.

Просмотрев код этого класса, вы получите представление о базовой архитектуре любой XNA-игры. Вам предстоит реализовать пять методов, перечисленных в табл. 1.1.

Таблица 1.1. Основные методы XNA

Метод	Описание
Initialize	Настроить все не связанное с содержимым игры.
LoadContent	Загрузить звуки, музыку, графику, шрифты и все остальное.
UnloadContent	Выгрузить все, не являющееся частью ContentManager (см. раздел «Код»).
Update	Реализовать логику игры – обработку нажатий клавиш, считывание состояния пульта, перемещение спрайтов и т. д.
Draw	Вывод на экран.

Обсудим некоторые методы более подробно.

`LoadContent`

Этот метод служит для загрузки содержимого. Под содержимым понимается все – от графики до звука, от музыки до шрифтов – вообще все, что поддерживает *конвейер содержимого* (Content Pipeline). Метод вызывается всякий раз, как системе требуется актив, который в данный момент отсутствует в памяти; возможно, более одного раза за игру.

В простейшем случае *конвейер содержимого* принимает на входе активы (графику, звуки, шрифты и т. д.), включенные в проект, и на этапе компиляции преобразует их в специальные файлы, которые среда исполнения XNA, работающая на ПК, Xbox 360 или Zune, может загрузить непосредственно. Это означает, что вам не нужно думать о написании конвертеров изображений, классов для импорта аудиофайлов и т. п. Достаточно перетащить мышью игровые активы в папку *Content*, которая является частью проекта, и интегрированная среда XNA возьмет на себя все остальное.

В данной игре будет содержимое трех типов: графика, звуки и шрифты. По ходу дела мы обсудим, как они загружаются, какими объектами представлены и как с ними работать.

В коде этого метода вы увидите, как создается объект SpriteBatch. Он предназначен для рисования группы спрайтов[1] с одинаковыми параметрами. Как этот объект используется, станет ясно позже, когды мы приступим к реализации методов рисования.

Update

Этот метод служит для обновления состояния игры. По умолчанию он вызывается 60 раз в секунду. Именно здесь обрабатывается ввод от пользователя, изменяются положения спрайтов, включается искусственный интеллект и т. д. Иными словами, в этом методе делается все, не требующее рисования.

Draw

В этом методе производится вывод на экран. По умолчанию он вызывается 60 раз в секунду на ПК и Xbox 360 и 30 раз раз в секунду на Zune. Отметим, однако, что между вызовами методов Update и Draw не обязательно существует взаимно-однозначное соответствие. В нашей двумерной игре мы будем пользоваться вышеупомянутым объектом SpriteBatch для рисования спрайтов в позициях, вычисленных в методе Update.

Экраны

В этой игре будет два «экрана»: экран заставки и экран игры. Чтобы выделить логику, необходимую для разработки экранов, мы создадим интерфейс IScreen, который будет реализован двумя классами: TitleScreen и GameScreen. Заведем перечисление GameState, чтобы можно было отслеживать, какой экран сейчас отображается. Его можно определить в файле *AllienAttackGame.cs* перед классом AllienAttackGame, как показано в примере 1.1.

Пример 1.1. Перечисление GameState

```
public enum GameState
{
  TitleScreen,
  GameScreen
};
```

Далее, чтобы не разводить беспорядок, создайте в проекте папку с именем *Screens*. Для этого щелкните правой кнопкой мыши по корню проекта на панели Solution Explorer и выберите из меню команду Add→New Folder. В папку добавьте новый интерфейс с именем IScreen. Для этого щелкните правой кнопкой мыши по папке *Screens*, выберите из меню

[1] Спрайт (англ. Sprite – фея; эльф) – графический объект в компьютерной графике. Чаще всего это растровое изображение, свободно перемещающееся по экрану. Один из важнейших графических элементов двухмерных компьютерных игр. – *Википедия.*

команду Add→New Item..., выберите пункт Interface на панели Templates (Шаблоны) и введите имя файла *IScreen.cs*. Классы обоих экранов будут реализовывать этот интерфейс, что позволит работать с ними единообразно. Код интерфейса представлен в примере 1.2.

Пример 1.2. Интерфейс IScreen

```
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Graphics;

namespace AlienAttack
{
  public interface IScreen
  {
    GameState Update(GameTime gameTime);
    void Draw(GameTime gameTime, SpriteBatch spriteBatch);
  }
}
```

Обратите внимание на два метода, которые кажутся очень знакомыми: `Update` и `Draw`. Методы с такими же именами, но слегка отличающимися сигнатурами, есть и в классе `AlienAttackGame`. Мы будем вызывать их для объекта экрана во время периодических циклов обновления и рисования игры. Иными словами, выполнение обновления и рисования будет делегировано какому-то из этих методов в зависимости от того, какой экран в данный момент отображается.

Экран заставки

Реализация экрана заставки несложна. Нужно лишь нарисовать логотип, вывести сообщение, предлагающее пользователю нажать клавишу, чтобы начать игру, и обработать нажатие клавиши.

Содержимое. Прежде всего, добавим содержимое для этого экрана. Создайте в папке *Content* новую папку *gfx*. В ней будет храниться вся относящаяся к игре графика. Из загруженных ранее игровых активов перетащите файлы *bgScreen.png* и *titleScreen.png* из папки *gfx* архива в только что созданную папку *gfx*. Затем перетащите файл *Arial.spritefont* из корневой папки архива в саму папку *Content*.

PNG – это один из графических форматов, который поддерживает канал прозрачности, что позволяет рисовать изображения, не задумываясь о фоне.

Файл *SpriteFont* заслуживает отдельного разговора. XNA содержит методы для вывода текста на экран, но нуждается в объекте `SpriteFont`, который знает, как выполнять рендеринг шрифта. *Arial.spritefont* – это XML-файл, в котором определены характеристики шрифта. *Конвейер содержимого* считывает этот файл и на этапе компиляции создает специальный графический файл, содержащий затребованные алфавитно-цифровые символы, нарисованные указанным шрифтом с учетом заданного стиля. Вы можете открыть файл и посмотреть, что в нем

находится. Все параметры очень хорошо документированы. Никто не мешает создать и свой собственный файл *SpriteFont*; для этого нужно щелкнуть правой кнопкой мыши по папке *Content*, выбрать из меню команду Add→New Item, а затем пункт Sprite Font.

Код. Теперь можно приступать к реализации экрана заставки. Создайте в папке *Screens* новый класс, назвав его `TitleScreen`. Он должен реализовывать интерфейс `IScreen`, то есть предоставлять реализацию обоих объявленных в нем методов `Update` и `Draw`. В примере 1.3 представлена заготовка этого класса.

Пример 1.3. Заготовка класса TitleScreen

```
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Graphics;
using Microsoft.Xna.Framework.Content;

namespace AlienAttack
{
  public class TitleScreen : IScreen
  {
    public TitleScreen(ContentManager contentManager)
    {
    }

    public GameState Update(GameTime gameTime)
    {
    }

    public void Draw(GameTime gameTime, SpriteBatch spriteBatch)
    {
    }
  }
}
```

Ранее вы добавили три актива для менеджера содержимого. Чтобы загрузить их в игру, понадобятся переменные, в которых будут храниться ссылки на объекты. Позже мы узнаем, что PNG-файлам соответствуют объекты типа `Texture2D`, а шрифту – объект типа `SpriteFont`. Поэтому объявим для их хранения переменные, как показано в примере 1.4.

Пример 1.4. Переменные-члены класса TitleScreen

```
public class TitleScreen : IScreen
{
  private Texture2D titleScreen;
  private Texture2D bgScreen;
  private SpriteFont arialFont;
```

Далее необходимо загрузить в эти переменные содержимое. Этим займется конструктор класса `TitleScreen`. Вставьте в него код, приведенный в примере 1.5.

Пример 1.5. Конструктор класса TitleScreen

```
public TitleScreen(ContentManager contentManager)
{
  titleScreen =
    contentManager.Load<Texture2D>("gfx\\titleScreen");
  bgScreen =
    contentManager.Load<Texture2D>("gfx\\bgScreen");
  arialFont = contentManager.Load<SpriteFont>("Arial");
}
```

Как видите, конструктор принимает объект типа `ContentManager`. Именно этот объект отвечает за загрузку во время выполнения содержимого, созданного конвейером. Метод `Load` является универсальным, при его конкретизации необходимо указать тип загружаемого объекта (в данном случае `Texture2D` и `SpriteFont`), а его первым аргументом служит путь к содержимому. По умолчанию конвейер помещает содержимое туда же, где находился исходный актив, и присваивает файлу то же самое имя, но без расширения.

Когда этот экран будет создан игровым движком, вы увидите, откуда берется объект `ContentManager`.

Теперь содержимое загружено и можно заняться реализацией методов `Update` и `Draw`. Сначала напишем метод `Update` (см. пример 1.6).

Пример 1.6. Метод Update в классе TitleScreen

```
public GameState Update(GameTime gameTime)
{
  if(InputManager.ControlState.Start)
    return GameState.GameScreen;
  return GameState.TitleScreen;
}
```

Мы просто используем класс `InputManager` (см. раздел «Класс InputManager» ниже) и, если выясняется, что нажата кнопка Start, то возвращаем элемент `GameScreen` перечисления `GameState`, сообщая управляющей программе, что следует изменить состояние. В противном случае мы возвращаем состояние `TitleScreen`.

Прежде чем приступить к рассмотрению класса `InputManager`, напишем метод `Draw` и покончим с реализацией этого класса. Код метода приведен в примере 1.7.

Пример 1.7. Метод Draw в классе TitleScreen

```
public void Draw(GameTime gameTime, SpriteBatch spriteBatch)
{
  spriteBatch.Draw(bgScreen, Vector2.Zero, Color.White);
  spriteBatch.Draw(titleScreen, Vector2.Zero, Color.White);
#if WINDOWS
  spriteBatch.DrawString( arialFont, "Press Enter or Start to Play",
                          new Vector2(600, 680), Color.White);
```

```
#endif

#if XBOX
  spriteBatch.DrawString( arialFont, "Press Start to Play",
                          new Vector2(600, 680),
Color.White);
#endif

#if ZUNE
  spriteBatch.DrawString( arialFont, "Press Play to Play",
                          new Vector2(80, 290),
Color.White);
#endif
}
```

В этом методе вышеупомянутый объект `SpriteBatch` применяется для рисования трех вещей: фоновой картинки, изображения заставки и текста в нижней части экрана с объяснением того, как начать игру.

При каждом обращении к методу `Draw` объекта `SpriteBatch` требуется передать объект типа `Texture2D` (он был загружен ранее), позицию, в которой его нужно нарисовать, и цвет. В первой строке рисуется фоновая картинка (рис. 1.3) – в позиции (0,0), которую как раз и представляет значение *Vector2.Zero*, и цветом White (то есть без цвета).

Во второй строке делается то же самое, только на этот раз поверх предыдущей картинки рисуется заставка `titleScreen`, показанная на рис. 1.4.

Поскольку в нужных местах заставка прозрачна, она отлично ложится на фоновое изображение.

Рис. 1.3. *Фоновая картинка*

***Рис. 1.4.** Заставка*

И наконец, мы выводим поверх обоих изображений текст, причем в различных позициях. Обратите внимание на директивы компилятора `WINDOWS`, `XBOX` и `ZUNE`, которые управляют выбором точки, с которой начинается вывод текста, в зависимости от платформы. Поскольку экран Zune гораздо меньше, чем для других устройств, необходимо выводить текст в другом месте, чтобы он располагался правильно. В результате всех этих действий на экране получается изображение, показанное на рис. 1.5.

***Рис. 1.5.** Окончательный вид экрана заставки*

Класс InputManager

Прежде чем двигаться дальше, обсудим упоминавшийся выше класс `InputManager`. Это довольно простой вспомогательный класс, который управляет вводом с клавиатуры, считыванием состояния пульта Xbox 360 и Zune – все в одном флаконе. Для выбора устройства, соответствующего платформе, необходимо снова воспользоваться директивами компилятора.

Создайте новый класс с именем `InputManager` в корне проекта. Это будет открытый статический класс, поэтому не забудьте добавить в определение ключевые слова `public static`. Замените вставленные при создании заготовки класса предложения `using` теми, что показаны в примере 1.8.

Пример 1.8. Предложения using в классе InputManager

```
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Input;
```

В самом начале файла, перед определением класса, создайте структуру с именем `ControlState`, как показано в примере 1.9.

Пример 1.9. Структура ControlState

```
public struct ControlState
{
  public bool Left;
  public bool Right;
  public bool Start;
  public bool Quit;
  public bool Fire;
}
```

В этой структуре пять элементов, описывающих возможные действия игрока. Корабль игрока может двигаться влево и вправо, а также стрелять. На глобальном уровне можно начать или прекратить игру.

Далее мы должны прочитать текущее состояние клавиатуры или пульта управления и сохранить ссылки на их состояние в предыдущем кадре, чтобы можно было понять, нажата ли клавиша только что или была нажата и раньше. Нужно это прежде всего потому, что мы хотим стрелять только один раз, а не в каждом кадре, где игрок держит клавишу стрельбы нажатой.

Добавьте переменные-члены, как показано в примере 1.10.

Пример 1.10. Переменные-члены класса InputManager

```
#if !ZUNE
  private static KeyboardState keyboardState, lastKeyboard;
#endif
  private static GamePadState gamePadState, lastGamePad;
  private static ControlState controlState;
```

Теперь напишем метод Update, который будет вызываться один раз в каждом кадре из метода Update игры, описанного выше. Код приведен в примере 1.11.

Пример 1.11. Метод Update класса InputManager

```
public static void Update()
{
#if !ZUNE
  keyboardState = Keyboard.GetState();
#endif

  gamePadState = GamePad.GetState(PlayerIndex.One);

  controlState.Quit  = (gamePadState.Buttons.Back == ButtonState.Pressed);
  controlState.Start = (gamePadState.Buttons.B    == ButtonState.Pressed);
  controlState.Left  = (gamePadState.DPad.Left    == ButtonState.Pressed);
  controlState.Right = (gamePadState.DPad.Right   == ButtonState.Pressed);
  controlState.Fire  = (gamePadState.Buttons.B    == ButtonState.Pressed &&
                        lastGamePad.Buttons.B     == ButtonState.Released);

#if !ZUNE
  controlState.Quit  = (controlState.Quit   ||
                        keyboardState.IsKeyDown(Keys.Escape));
  controlState.Start = (controlState.Start  ||
                        keyboardState.IsKeyDown(Keys.Enter) ||
                        gamePadState.IsButtonDown(Buttons.Start));
  controlState.Left  = (controlState.Left   ||
                        gamePadState.ThumbSticks.Left.X < -0.1f);
  controlState.Right = (controlState.Right  ||
                        gamePadState.ThumbSticks.Left.X > 0.1f);
  controlState.Left  = (controlState.Left   ||
                        keyboardState.IsKeyDown(Keys.Left));
  controlState.Right = (controlState.Right  ||
                        keyboardState.IsKeyDown(Keys.Right));
  controlState.Fire  = (controlState.Fire   ||
                        keyboardState.IsKeyDown(Keys.Space) &&
                        !lastKeyboard.IsKeyDown(Keys.Space));
#endif

  lastGamePad = gamePadState;

#if !ZUNE
  lastKeyboard = keyboardState;
#endif
}
```

Здесь мы получаем и запоминаем состояние клавиатуры и состояние первого пульта. Далее с помощью объектов GamePadState и KeyboardState опрашивается состояние различных клавиш и полям структуры controlState присваиваются те или иные булевские значения в зависимости от того, что нажато, а что нет. Наконец, текущие значения сохраняются для использования в следующем кадре, чтобы можно было понять, нажата кнопка только что или удерживается.

Обратите также внимание на использование директив компилятора, которые говорят, какие участки программы компилировать для конкретной платформы. У плеера Zune нет клавиатуры, зато есть кнопки, которые опрашивает объект `GamePadState`. К пульту приставки Xbox 360 можно подключить клавиатуру (устройство *Chatpad*) и считывать нажатия клавиш точно так же, как для клавиатуры обычного ПК.

Напоследок добавим свойство для возврата текущего значения переменной-члена `controlState` (пример 1.12).

Пример 1.12. Свойство ControlState

```
public static ControlState ControlState
{
  get { return controlState; }
}
```

Назад к игре

Итак, у нас есть объект, представляющий экран заставки, активы для него и способ обрабатывать действия пользователя. Теперь можно подключить экран заставки к главному классу игры – пусть обрабатывает и рисует.

Вернитесь к классу `AlienAttackGame` и добавьте переменные-члены, как показано в примере 1.13. Обратите внимание, что с помощью директивы компилятора `#if` мы специально устанавливаем размер экрана для Zune.

Пример 1.13. Переменные-члены класса AlienAttackGame

```
  private GameState gameState;
  private IScreen screen;
#if ZUNE
  public static int ScreenWidth = 240;
  public static int ScreenHeight = 320;
#else
  public static int ScreenWidth = 1024;
  public static int ScreenHeight = 768;
#endif
```

В переменной `gameState` хранится текущее состояние игры (какой экран сейчас отображается), а в переменной `screen` – экземпляр класса, реализующего интерфейс `IScreen`, в данном случае нашего класса `TitleScreen`, а в скором времени и класса `GameScreen`.

Переменные `ScreenWidth` и `ScreenHeight` содержат размер экрана, на котором будет разворачиваться игра. Для ПК и Xbox 360 по умолчанию принимается размер 1024×768. Собственный экран Zune имеет разрешение 240×320, оно и принимается по умолчанию. Отметим, что обе переменные открытые и статические, чтобы к ним можно было легко обратиться из любого места программы.

Далее необходимо сообщить XNA о том, что игра будет вестись на экране с заданным разрешением. Для этого устанавливаются свойства PreferredBackBufferWidth и PreferredBackBufferHeight объекта GraphicsDevice, созданного в конструкторе (пример 1.14).

Пример 1.14. Задание размера экрана в конструкторе AlienAttackGame

```
public AlienAttackGame()
{
  graphics = new GraphicsDeviceManager(this);

  // задаем размер экрана в зависимости от устройства
  graphics.PreferredBackBufferWidth = ScreenWidth;
  graphics.PreferredBackBufferHeight = ScreenHeight;

  Content.RootDirectory = "Content";
}
```

Теперь в методе Initialize создадим объект класса TitleScreen и установим начальное состояние игры (пример 1.15).

Пример 1.15. Метод Initialize

```
protected override void Initialize()
{
  // создаем экран заставки
  screen = new TitleScreen(this.Content);
  gameState = GameState.TitleScreen;

  base.Initialize();
}
```

Далее необходимо сделать так, чтобы метод игры Update обрабатывал действия пользователя с помощью созданного выше класса InputManager, а затем обновлял объект IScreen (в котором сейчас находится объект нашего класса TitleScreen). Обновленный код метода Update приведен в примере 1.16.

Пример 1.16. Метод Update

```
protected override void Update(GameTime gameTime)
{
  // обработать пользовательский ввод
  InputManager.Update();

  // проверить, не пора ли завершать игру
  if(InputManager.ControlState.Quit)
    this.Exit();

  // обновить состояние экрана
  GameState newState = screen.Update(gameTime);

  base.Update(gameTime);
}
```

Обратите внимание, что в этом методе мы спрашиваем у класса `InputManager`, была ли нажата кнопка Quit (Выйти), и, если да, немедленно завершаем программу.

И наконец, в методе `Draw` класса `AlienAttackGame` мы рисуем экран заставки. Обновленный код этого метода приведен в примере 1.17.

Пример 1.17. Метод Draw

```
protected override void Draw(GameTime gameTime)
{
  // открыть spritebatch, нарисовать экран, закрыть
  this.spriteBatch.Begin();
      screen.Draw(gameTime, this.spriteBatch);
  this.spriteBatch.End();
}
```

Вызывая метод `Begin` объекта `spriteBatch`, мы сообщаем ему, что готова группа спрайтов с одинаковыми параметрами. Затем мы поручаем рисование объекту экрана, вызывая его метод `Draw`, и в конце закрываем объект `spriteBatch`, вызывая его метод `End`.

Полюбуемся на заставку

Итак, все на месте, и настало время взглянуть, что получилось. Откомпилировав и запустив приложение, вы увидите картину, изображенную на рис. 1.6.

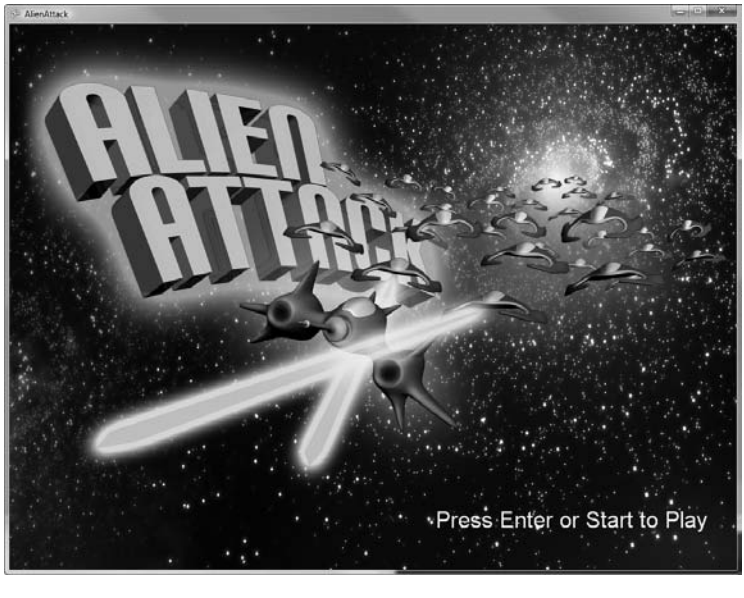

***Рис. 1.6.** Экран заставки*

Экран игры

Покончив с экраном заставки, мы можем приступить к реализации самой игры. Для начала добавьте новый класс с именем `GameScreen` в папку *Screens*. Как и `TitleScreen`, этот класс будет реализовывать интерфейс `IScreen`, то есть предоставлять методы `Update` и `Draw`. И, как и раньше, нам прежде всего нужно нарисовать фон со звездами. Код практически такой же, как для экрана заставки; первый вариант класса `GameScreen` представлен в примере 1.18.

Пример 1.18. Заготовка класса GameScreen

```
using System.Collections.Generic;
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Graphics;
using Microsoft.Xna.Framework.Content;

namespace AlienAttack
{
  public class GameScreen : IScreen
  {
    private ContentManager contentManager;
    private Texture2D bgScreen;

    public GameScreen(ContentManager cm)
    {
      contentManager = cm;
      bgScreen = contentManager.Load<Texture2D>("gfx\\bgScreen");
    }

    public GameState Update(GameTime gameTime)
    {
      return GameState.GameScreen;
    }

    public void Draw(GameTime gameTime, SpriteBatch spriteBatch)
    {
      spriteBatch.Draw(bgScreen, Vector2.Zero, Color.White);
    }
  }
}
```

Включение GameScreen в код программы

Подключить этот класс к основному коду игры совсем просто. В классе `AlienAttackGame` измените метод `Update`, как показано в примере 1.19. Новые и измененные строки выделены полужирным шрифтом, чтобы их сразу было видно (так мы будем поступать и впредь).

Пример 1.19. Модифицированный метод Update

```
protected override void Update(GameTime gameTime)
{
  // обработать пользовательский ввод
```

```
    InputManager.Update();

    // проверить, не пора ли завершать игру
    if(InputManager.ControlState.Quit)
      this.Exit();

    // обновить состояние экрана
    GameState newState = screen.Update(gameTime);

    // если экран сменился, обработать изменение
    if(gameState != newState)
    {
      switch(newState)
      {
        case GameState.TitleScreen:
          screen = new TitleScreen(this.Content);
          break;
        case GameState.GameScreen:
          screen = new GameScreen(this.Content);
          break;
      }
      gameState = newState;
    }

    base.Update(gameTime);
  }
```

В этой версии метода мы проверяем значение, возвращенное методом экрана `Update`. Если полученное состояние отличается от текущего, то мы создаем экран, соответствующий новому состоянию, и продолжаем работать.

Библиотека работы со звуком

В состав выпуска XNA 3.0 входит совершенно новый API для работы со звуком, который гораздо проще существующего XACT API. Новая библиотека позволяет безо всякого труда воспроизводить простые звуковые эффекты, а для этой игры больше ничего и не требуется.

Перетащите папку *sfx* из загруженного архива с активами в конвейер содержимого. Затем создайте класс `AudioManager` в корне проекта. Мы определим перечисление, содержащее все имеющиеся звуковые эффекты, загрузим их в конструкторе класса `AudioManager` и напишем метод `PlayCue`, который будет воспроизводить запрошенный эффект. Все это показано в примере 1.20.

Пример 1.20. Класс AudioManager

```
using Microsoft.Xna.Framework.Audio;
using Microsoft.Xna.Framework.Content;

namespace AlienAttack
{
  public class AudioManager
```

```
  {
    // все имеющиеся звуковые эффекты
    public enum Cue
    {
      Theme,
      EnemyShot,
      PlayerShot,
      Explosion
    };

    // экземпляры этих эффектов
    private SoundEffect theme;
    private SoundEffect enemyShot;
    private SoundEffect playerShot;
    private SoundEffect explosion;

    public AudioManager(ContentManager contentManager)
    {
      // загружаем их
      theme = contentManager.Load<SoundEffect>("sfx\\theme");
      enemyShot = contentManager.Load<SoundEffect>("sfx\\enemyShot");
      playerShot = contentManager.Load<SoundEffect>("sfx\\playerShot");
      explosion = contentManager.Load<SoundEffect>("sfx\\explosion");
    }

    public void PlayCue(Cue cue)
    {
      // воспроизводим запрошенный эффект
      switch(cue)
      {
        case Cue.Theme:
          theme.Play();
          break;
        case Cue.EnemyShot:
          enemyShot.Play();
          break;
        case Cue.PlayerShot:
          playerShot.Play();
          break;
        case Cue.Explosion:
          explosion.Play();
          break;
      }
    }
  }
}
```

Теперь можно добавить создание объекта `AudioManager` **в класс** `AlienAttackGame`**. Мы сделаем этот объект статическим, чтобы к нему можно было обращаться из любого класса. Добавьте статическую переменную-член типа** `AudioManager` **в класс** `AlienAttackGame` **и инициализируйте ее в методе** `Initialize`**, как показано в примере 1.21.**

Пример 1.21. Создание AudioManager

```
public static AudioManager AudioManager;

protected override void Initialize()
{
  // создаем экран заставки
  screen = new TitleScreen(this.Content);
  gameState = GameState.TitleScreen;

  // создаем вспомогательный объект для работы со звуком
  AudioManager = new AudioManager(this.Content);

  base.Initialize();
}
```

Этим объектом можно сразу же воспользоваться для проигрывания фоновой музыки, нужно лишь добавить вызов метода `PlayCue` в конструктор `GameScreen`, как показано в примере 1.22.

Пример 1.22. Проигрывание фоновой музыки

```
public GameScreen(ContentManager cm)
{
  contentManager = cm;
  AlienAttackGame.AudioManager.PlayCue(AudioManager.Cue.Theme);
  bgScreen = contentManager.Load<Texture2D>("gfx\\bgScreen");
}
```

Корабль игрока и спрайты

Теперь нужно поместить на экран игры корабль игрока. Он будет располагаться внизу экрана и перемещаться влево и вправо. Первым делом перетащите файл *player.png* из ранее загруженных активов в папку *gfx*. В результате он будет включен в конвейер содержимого, как и предыдущие изображения.

Спрайтом называется любой графический элемент, рисуемый на экране, и корабль игрока – один из многих спрайтов, которые еще встретятся нам в игре. Нам предстоит нарисовать и этот корабль, и врагов, и выстрелы нашего корабля, и выстрелы врагов, так что имеет смысл завести класс `Sprite`, которому все эти объекты будут наследовать.

Этот класс будет отвечать за загрузку изображения одного спрайта, за позиционирование его на экране, за скорость перемещения и несколько дополнительных свойств, определяющих ширину, высоту и охватывающий прямоугольник спрайта.

Создайте в проекте новую папку *Sprites*. В нее добавьте класс `Sprite` и реализуйте его, как показано в примере 1.23.

Пример 1.23. Реализация класса Sprite

```
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Content;
```

```
using Microsoft.Xna.Framework.Graphics;

namespace AlienAttack
{
  public abstract class Sprite
  {
    // все текстуры в наборе анимаций
    protected Texture2D[] spriteTextures;

    // текущий рисуемый кадр
    protected int frameIndex;

    public ContentManager Content;
    public Vector2 Position;
    public Vector2 Velocity;

    // охватывающий прямоугольник изображения...
    // нужен для обнаружения столкновений
    private Rectangle boundingBox;

    public Sprite()
    {
    }

    public Sprite(ContentManager contentManager)
    {
      this.Content = contentManager;
    }

    public Sprite(ContentManager contentManager, string contentName) :
        this(contentManager)
    {
      spriteTextures = new Texture2D[1];

      // загрузить изображение
      spriteTextures[0] = this.Content.Load<Texture2D>(contentName);
    }

    public virtual void Update(GameTime gameTime)
    {
      // переместить спрайт с учетом заданной скорости
      this.Position += this.Velocity;
    }

    public virtual void Draw(GameTime gameTime, SpriteBatch spriteBatch)
    {
      spriteBatch.Draw(spriteTextures[frameIndex], this.Position,
                       Color.White);
    }

    public virtual int Width
    {
      get { return spriteTextures[0].Width; }
    }

    public virtual int Height
```

```
    {
      get { return spriteTextures[0].Height; }
    }

    public virtual Rectangle BoundingBox
    {
      get
      {
        // это присваивание необходимо только один раз
        if(boundingBox == Rectangle.Empty)
        {
          boundingBox.Width = this.Width;
          boundingBox.Height = this.Height;
        }
        boundingBox.X = (int)this.Position.X;
        boundingBox.Y = (int)this.Position.Y;

        return boundingBox;
      }
    }
  }
}
```

Этот класс представляет очень простой спрайт. У него есть положение на экране, скорость перемещения, ширина, высота и охватывающий прямоугольник, зависящий от текущей позиции (см. ниже раздел «Обнаружение столкновений и взрывы»). Кроме того, класс содержит массив подлежащих загрузке изображений `spriteTextures` (объектов класса `Texture2D`). Это все кадры анимации, необходимые для рисования спрайта на экране. В переменной `frameIndex` хранится индекс текущего кадра, алгоритм его обновления будет реализован в производных классах.

Имея базовый класс, мы можем реализовать объект, представляющий корабль игрока.

Добавьте в папку `sprites` новый класс `Player` и реализуйте его, как показано в примере 1.24.

Пример 1.24. Реализация класса Player

```
using Microsoft.Xna.Framework.Content;

namespace AlienAttack
{
  public class Player : Sprite
  {
    public Player(ContentManager contentManager) : base(contentManager,
          "gfx\\player")
    {
      this.Position.X = AlienAttackGame.ScreenWidth/2 - this.Width/2;
#if ZUNE
      this.Position.Y = AlienAttackGame.ScreenHeight - 40;
```

```
#else
      this.Position.Y = AlienAttackGame.ScreenHeight - 100;
#endif
    }
  }
}
```

Этот класс просто загружает графику для игрока (спрайт с одним кадром) и устанавливает позицию по умолчанию в нижней части экрана посередине. Для обновления и рисования спрайта используются методы `Update` и `Draw`, унаследованные от базового класса `Sprite`.

Возвращаясь к классу `GameScreen`, мы можем создать объект `Player`, обновить и нарисовать его. Для этого придется добавить в класс `GameScreen` новые переменные-члены, изменить конструктор и методы `Update` и `Draw`, а также включить новый метод `MovePlayer` (пример 1.25).

Пример 1.25. Добавление игрока в класс GameScreen

```
private Player player;
private float PlayerVelocity = AlienAttackGame.ScreenWidth / 200.0f;

public GameScreen(ContentManager cm)
{
  contentManager = cm;
  AlienAttackGame.AudioManager.PlayCue(AudioManager.Cue.Theme);
  bgScreen = contentManager.Load<Texture2D>("gfx\\bgScreen");
  player = new Player(contentManager);
}

public GameState Update(GameTime gameTime)
{
  MovePlayer(gameTime);
  return GameState.GameScreen;
}

public void Draw(GameTime gameTime, SpriteBatch spriteBatch)
{
  spriteBatch.Draw(bgScreen, Vector2.Zero, Color.White);

  // нарисовать игрока
  if(player != null)
    player.Draw(gameTime, spriteBatch);
}

private void MovePlayer(GameTime gameTime)
{
  if(player != null)
  {
    // сдвинуть влево
    if(InputManager.ControlState.Left && player.Position.X > 0)
      player.Position.X -= PlayerVelocity;

    // сдвинуть вправо
    if(InputManager.ControlState.Right &&
```

```
          player.Position.X + player.Width < AlienAttackGame.ScreenWidth)
        player.Position.X += PlayerVelocity;
        player.Update(gameTime);
      }
    }
```

Отметим, что метод `Update` пользуется ранее написанным объектом `InputManager` для проверки состояния кнопок `Left` (влево) и `Right` (вправо). Если та или другая нажата, то корабль перемещается в соответствующем направлении.

Если вы сейчас запустите игру, то услышите фоновую музыку, увидите фоновую картину и одинокий корабль внизу экрана, который перемещается влево и вправо в ответ на нажатия клавиш со стрелками или кнопок на пульте. Интересной такую игру не назовешь, но для начала уже неплохо.

Выстрелы игрока

Теперь позволим игроку стрелять. Начнем с добавления новых файлов в конвейер содержимого. Перетащите папку *pshot* из загруженных активов в папку *gfx* внутри проекта. В результате появится новый каталог и его содержимое: два файла с изображениями, которые составят короткую анимацию выстрела.

Затем создадим в папке *Sprites* новый класс `PlayerShot`, наследующий `Sprite` (пример 1.26).

Пример 1.26. Класс PlayerShot

```
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Content;
using Microsoft.Xna.Framework.Graphics;

namespace AlienAttack
{
  public class PlayerShot : Sprite
  {
    // все кадры анимации
    private double lastTime;

    public PlayerShot(ContentManager contentManager) : base(contentManager)
    {
      spriteTextures = new Texture2D[2];

      // загружаем кадры
      for(int i = 0; i <= 1; i++)
        spriteTextures[i] =
          contentManager.Load<Texture2D>("gfx\\pshot\\pshot_" + i);
    }

    public override void Update(GameTime gameTime)
    {
      // перерисовываем кадр каждые 200 мс... похоже, этого достаточно
```

```
        if(gameTime.TotalGameTime.TotalMilliseconds - lastTime > 200)
        {
          // переключаемся между кадрами
          frameIndex = frameIndex == 0 ? 1 : 0;
          lastTime = gameTime.TotalGameTime.TotalMilliseconds;
        }
        this.Position.Y -= 5;
      }
    }
  }
```

Конструктор этого класса загружает и сохраняет два кадра анимации. Они позволяют придать перемещающемуся по экрану спрайту эффект вспышки. Метод `Update` следит за временем и по истечении 200 миллисекунд рисует следующий кадр. За сам процесс рисования отвечает базовый класс `Sprite`.

Теперь мы можем добавить в класс `GameScreen` код, который позволит игроку стрелять. Поскольку одновременно на экране может быть видно несколько выстрелов, будем хранить все экземпляры в объекте типа `List`. В методе `Update` мы будем вызывать новый метод `UpdatePlayerShots`, который проверяет, была ли нажата кнопка `Fire` (Огонь), и, если да и при этом прошло определенное время (мы не хотим создавать выстрелы в каждом кадре), то создает новый объект `PlayerShot`, добавляет его в список и воспроизводит подходящий звуковой эффект. В конце метод `Update` обходит список выстрелов игрока, обновляет их позиции и удаляет те выстрелы, которые оказались за пределами экрана. Вызываемый позже метод `Draw` обходит обновленный список выстрелов и рисует их на экране. Все это показано в примере 1.27.

Пример 1.27. Использование класса PlayerShot

```
private List<PlayerShot> playerShots;
private double lastTime;

public GameScreen(ContentManager cm)
{
  contentManager = cm;
  AlienAttackGame.AudioManager.PlayCue(AudioManager.Cue.Theme);
  bgScreen = contentManager.Load<Texture2D>("gfx\\bgScreen");
  player = new Player(contentManager);

  playerShots = new List<PlayerShot>();
}

public GameState Update(GameTime gameTime)
{
  MovePlayer(gameTime);
  UpdatePlayerShots(gameTime);

  return GameState.GameScreen;
}
```

```
public void Draw(GameTime gameTime, SpriteBatch spriteBatch)
{
  spriteBatch.Draw(bgScreen, Vector2.Zero, Color.White);

  // нарисовать игрока

  if(player != null)
    player.Draw(gameTime, spriteBatch);

  // нарисовать выстрелы игрока
  foreach(PlayerShot playerShot in playerShots)
    playerShot.Draw(gameTime, spriteBatch);
}

private void UpdatePlayerShots(GameTime gameTime)
{
  // если разрешено стрелять, добавить выстрел в список
  if( InputManager.ControlState.Fire &&
      gameTime.TotalGameTime.TotalMilliseconds - lastTime > 500)
  {
    // создать новый выстрел
    PlayerShot ps = new PlayerShot(this.contentManager);
    ps.Position.X = (player.Position.X + player.Width/2) - ps.Width/2;
    ps.Position.Y = player.Position.Y - ps.Height;
    playerShots.Add(ps);
    lastTime = gameTime.TotalGameTime.TotalMilliseconds;
    AlienAttackGame.AudioManager.PlayCue(AudioManager.Cue.PlayerShot);
  }

  // обойти выстрелы, присутствующие на экране
  for(int i = 0; i < playerShots.Count; i++)
  {
    PlayerShot playerShot = playerShots[i];

    playerShot.Update(gameTime);

    // если выстрел вышел за пределы экрана, удалить его из списка
    if(playerShot.Position.Y + playerShot.Height < 0)
    {
      playerShots.RemoveAt(i);
      playerShot = null;
    }
  }
}
```

Запустив сейчас игру, вы получите возможность стрелять.

Подсчет очков и оставшихся жизней

Далее реализуем отображение набранных очков и оставшихся жизней в левом нижнем углу экрана. Для этого понадобится использованный ранее шрифт Arial, но, поскольку он уже помещен в конвейер содержимого, то добавлять его еще раз не нужно.

Набранные очки и оставшиеся жизни будут выводиться шрифтом Arial SpriteFont. Кроме того, мы воспользуемся уже созданным спрайтом `Player` для вывода значка корабля слева от счетчика оставшихся жизней. Код приведен в примере 1.28.

Пример 1.28. Подсчет очков и оставшихся жизней

```
private Player livesIcon;
private int score;
private int lives;
private SpriteFont arial;
private bool loseGame;

public GameScreen(ContentManager cm)
{
  contentManager = cm;
  AlienAttackGame.AudioManager.PlayCue(AudioManager.Cue.Theme);
  bgScreen = contentManager.Load<Texture2D>("gfx\\bgScreen");
  player = new Player(contentManager);

  playerShots = new List<PlayerShot>();

  arial = contentManager.Load<SpriteFont>("arial");

  // нарисовать в левом нижнем углу значок оставшихся жизней
  livesIcon = new Player(contentManager);

#if ZUNE
  livesIcon.Position = new Vector2(0, AlienAttackGame.ScreenHeight-20);
#else
  livesIcon.Position = new Vector2(40, AlienAttackGame.ScreenHeight-60);
#endif

  lives = 2;
}

public void Draw(GameTime gameTime, SpriteBatch spriteBatch)
{
  spriteBatch.Draw(bgScreen, Vector2.Zero, Color.White);

  // нарисовать игрока
  if(player != null)
    player.Draw(gameTime, spriteBatch);

  // нарисовать выстрелы игрока
  foreach(PlayerShot playerShot in playerShots)
    playerShot.Draw(gameTime, spriteBatch);

#if ZUNE
  // вывести набранные очки
  spriteBatch.DrawString(arial, "Score", new Vector2(0, 0), Color.White);
  spriteBatch.DrawString(arial, score.ToString(), new Vector2(0, 20),
  Color.White);

  // вывести значок с оставшимися жизнями
  livesIcon.Draw(gameTime, spriteBatch);
```

```
    spriteBatch.DrawString(arial, "x" + lives.ToString(),
        new Vector2(livesIcon.Position.X + livesIcon.Width + 4,
              livesIcon.Position.Y),
        Color.White);
#else
    // вывести набранные очки
    spriteBatch.DrawString(arial, "Score", new Vector2(50, 50), Color.White);
    spriteBatch.DrawString(arial, score.ToString(), new Vector2(50, 80),
    Color.White);

    // вывести значок с оставшимися жизнями
    livesIcon.Draw(gameTime, spriteBatch);
    spriteBatch.DrawString(arial, "x" + lives.ToString(),
        new Vector2(livesIcon.Position.X + livesIcon.Width + 4,
              livesIcon.Position.Y+8),
      Color.White);
#endif
}
```

Запустив программу, вы увидите количество набранных очков и оставшихся жизней и сможете стрелять, нажимая пробельную клавишу или кнопку A на пульте управления.

Теперь нам нужны враги.

Враги

Враги в нашей игре представляются еще одним спрайтом. Перетащите папку *enemy1* из загруженных активов в папку *gfx* проекта. Затем создайте в папке *Sprites* новый класс `Enemy`. Как и все прочие спрайты, он будет наследовать базовому классу `Sprite`. Реализация очень напоминает класс `PlayerShot`. Мы должны загрузить все кадры анимации и изменять индекс кадра в методе `Update`. Анимация заключается в обходе всего набора кадров сначала в прямом, а потом в обратном направлении. Код показан в примере 1.29.

Пример 1.29. Реализация класса Enemy

```
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Content;
using Microsoft.Xna.Framework.Graphics;

namespace AlienAttack
{
  public class Enemy : Sprite
  {
    // в каком направлении мы сейчас обходим набор кадров?
    private int direction = 1;

    double lastTime;

    public Enemy(ContentManager contentManager)
    {
      spriteTextures = new Texture2D[10];
```

```
        // загружаем текстуры spriteTextures
        for(int i = 0; i <= 9; i++)
          spriteTextures[i] =
            contentManager.Load<Texture2D>("gfx\\enemy1\\enemy1_" + i);
        this.Velocity.X = 1;
      }

      public override void Update(GameTime gameTime)
      {
        // если мы достигли конца анимации, меняем направление
        if(frameIndex == 9)
          direction = -1;

        // если мы достигли начала анимации, меняем направление
        if(frameIndex == 0)
          direction = 1;

        // через каждые 70 мс переходим к очередному кадру
        if(gameTime.TotalGameTime.TotalMilliseconds - lastTime > 70)
        {
          frameIndex += direction;
          lastTime = gameTime.TotalGameTime.TotalMilliseconds;
        }
      }
    }
  }
```

Группа врагов

Выше на снимке с экрана было видно, что по экрану перемещаются враги, расположенные в узлах прямоугольной матрицы. Чтобы было проще управиться со всей группой, создадим еще один производный от `Sprite` класс под названием `EnemyGroup`. Этот класс будет отвечать за создание матрицы врагов, перемещение их по экрану, пальбу в корабль игрока и обработку столкновений с выстрелами игрока.

Создайте в папке *Sprites* класс `EnemyGroup`, унаследовав его от `Sprite`, как любой другой спрайт. Он будет заметно отличаться от ранее созданных спрайтов, но все же позволит воспользоваться уже реализованной функциональностью, общей для всех спрайтов.

Этот класс довольно велик, поэтому я разобью его на части не совсем так, как в предыдущих разделах. Пока что нас будет интересовать только создание матрицы врагов и перемещение их по экрану. Начнем с заготовки класса, показанной в примере 1.30.

Пример 1.30. Заготовка класса EnemyGroup

```
using System;
using System.Collections.Generic;
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Content;
using Microsoft.Xna.Framework.Graphics;
```

```
namespace AlienAttack
{
  public class EnemyGroup : Sprite
  {
    // группа врагов
    private Enemy[,] enemies;

    // ширина одного врага
    private int enemyWidth;

#if ZUNE
    private const int EnemyRows = 4; // число строк в матрице
    private const int EnemyCols = 6; // число столбцов в матрице
    private const int EnemyVerticalJump = 3; // на сколько пикселов
                                             // опускаться по вертикали,
                                             // упершись в край
    private const int EnemyStartPosition = 10; // позиция в матрице
                                               // по вертикали
    private const int ScreenEdge = 3; // виртуальный край экрана
                                      // для смены направления
    private Vector2 EnemySpacing = new Vector2(2, 2); // расстояние между
                                                      // спрайтами
    private const float EnemyVelocity = 0.5f; // скорость перемещения
                                              // матрицы в одном кадре
#else
    private const int EnemyRows = 4; // число строк в матрице
    private const int EnemyCols = 8; // число столбцов в матрице
    private const int EnemyVerticalJump = 10; // на сколько пикселов
                                              // опускаться по вертикали,
                                              // упершись в край
    private const int EnemyStartPosition = 100; // позиция в матрице
                                                // по вертикали
    private const int ScreenEdge = 20; // виртуальный край экрана
                                       // для смены направления
    private Vector2 EnemySpacing = new Vector2(4, 4); // расстояние между
                                                      // спрайтами
    private const float EnemyVelocity = 1.5f; // скорость перемещения
                                              // матрицы в одном кадре
#endif

    public EnemyGroup(ContentManager contentManager) : base(contentManager)
    {
    }

    public override void Update(GameTime gameTime)
    {
    }

    public override void Draw(GameTime gameTime, SpriteBatch spriteBatch)
    {
    }
  }
}
```

Объявленные константы еще понадобятся нам по мере разработки класса.

Конструктор класса создает матрицу врагов и располагает ее в центре экрана (пример 1.31).

Пример 1.31. Конструктор класса EnemyGroup

```
public EnemyGroup(ContentManager contentManager) : base(contentManager)
{
  enemies = new Enemy[EnemyRows,EnemyCols];

  // создаем матрицу врагов
  for(int y = 0; y < EnemyRows; y++)
  {
    for(int x = 0; x < EnemyCols; x++)
    {
      Enemy enemy = new Enemy(contentManager);
      enemy.Position.X = x * enemy.Width + EnemySpacing.X;
      enemy.Position.Y = y * enemy.Height + EnemySpacing.Y;
      enemies[y,x] = enemy;
    }
  }

  enemyWidth = enemies[0,0].Width;

  // центрируем матрицу по вертикали
  this.Position.X = AlienAttackGame.ScreenWidth/2 -
        ((EnemyCols * (enemyWidth + EnemySpacing.X)) / 2);
  this.Position.Y = EnemyStartPosition;
  this.Velocity.X = EnemyVelocity;
}
```

Метод `Update`, показанный в примере 1.32, пользуется методом `MoveEnemies` для перемещения врагов влево и вправо по экрану, опуская их на несколько пикселов после соударения с краем. Мы напишем еще несколько вспомогательных методов для определения самого левого и самого правого из еще оставшихся в матрице врагов, чтобы знать, когда левая или правая сторона матрицы упирается в край экрана.

Пример 1.32. Метод Update в классе EnemyGroup

```
public override void Update(GameTime gameTime)
{
  MoveEnemies(gameTime);
}

private Enemy FindRightMostEnemy()
{
  // найти самого правого врага в матрице
  for(int x = EnemyCols-1; x > -1; x--)
  {
    for(int y = 0; y < EnemyRows; y++)
    {
```

```
      if(enemies[y,x] != null)
        return enemies[y,x];
    }
  }
  return null;
}

private Enemy FindLeftMostEnemy()
{
  // найти самого левого врага в матрице
  for(int x = 0; x < EnemyCols; x++)
  {
    for(int y = 0; y < EnemyRows; y++)
    {
      if(enemies[y,x] != null)
        return enemies[y,x];
    }
  }
  return null;
}

public bool AllDestroyed()
{
  // если врагов не осталось, мы победили
  return (FindLeftMostEnemy() == null);
}

private void MoveEnemies(GameTime gameTime)
{
  Enemy enemy = FindRightMostEnemy();

  // если самый правый враг уперся в край экрана,
     сменить направление движения
  if(enemy != null)
  {
    if(enemy.Position.X + enemy.Width >  AlienAttackGame.ScreenWidth -
          ScreenEdge)
    {
      this.Position.Y += EnemyVerticalJump;
      this.Velocity.X = -EnemyVelocity; // сдвинуть влево
    }
  }

  enemy = FindLeftMostEnemy();

  // если самый левый враг уперся в край экрана, сменить направление движения
  if(enemy != null)
  {
    if(enemy.Position.X < ScreenEdge)
    {
      this.Position.Y += EnemyVerticalJump;
      this.Velocity.X = EnemyVelocity; // сдвинуть вправо
    }
  }
```

```
    // пересчитать позиции врагов
    for(int y = 0; y < EnemyRows; y++)
    {
      for(int x = 0; x < EnemyCols; x++)
      {
        if(enemies[y,x] != null)
        {
          // X = позиция матрицы в целом +
          // (X внутри матрицы * ширина врага) + отступ
          // Y = позиция матрицы в целом +
          // (Y внутри матрицы * ширина врага) + отступ
          enemies[y,x].Position.X =
             (this.Position.X + (x * (enemyWidth + EnemySpacing.X)));
          enemies[y,x].Position.Y =
             (this.Position.Y + (y * (enemyWidth + EnemySpacing.X)));
          enemies[y,x].Update(gameTime);
        }
      }
    }

    this.Position += this.Velocity;
  }
```

Эти методы понадобятся нам позже, чтобы понять, уничтожены ли все враги и нужно ли перерисовывать игровое поле.

И наконец, нарисуем всех врагов в их текущих позициях (пример 1.33).

Пример 1.33. Метод Draw в классе EnemyGroup

```
public override void Draw(GameTime gameTime, SpriteBatch spriteBatch)
{
  // нарисовать всех активных врагов
  foreach(Enemy enemy in enemies)
  {
    if(enemy != null)
      enemy.Draw(gameTime, spriteBatch);
  }
}
```

Пора включать класс `EnemyGroup` в `GameScreen`. Создаем экземпляр `EnemyGroup` в конструкторе, обновляем его в методе `Update` и рисуем в методе `Draw`, как показано в примере 1.34.

Пример 1.34. Использование EnemyGroup в GameScreen

```
private EnemyGroup enemyGroup;

public GameScreen(ContentManager cm)
{
  contentManager = cm;
  AlienAttackGame.AudioManager.PlayCue(AudioManager.Cue.Theme);
  bgScreen = contentManager.Load<Texture2D>("gfx\\bgScreen");
```

```
  player = new Player(contentManager);

  playerShots = new List<PlayerShot>();

  arial = contentManager.Load<SpriteFont>("arial");

  // в левом нижнем углу рисуем оконку с количеством жизней
  livesIcon = new Player(contentManager);
#if ZUNE
  livesIcon.Position = new Vector2(0, AlienAttackGame.ScreenHeight-20);
#else
  livesIcon.Position = new Vector2(40, AlienAttackGame.ScreenHeight-60);
#endif

  enemyGroup = new EnemyGroup(contentManager);

  lives = 2;
}

public GameState Update(GameTime gameTime)
{
  MovePlayer(gameTime);
  HandlePlayerShots(gameTime);

  // если мы еще не проиграли, обновляем позиции врагов
  if(!loseGame)
    enemyGroup.Update(gameTime);

  return GameState.GameScreen;
}

public void Draw(GameTime gameTime, SpriteBatch spriteBatch)
{
  spriteBatch.Draw(bgScreen, Vector2.Zero, Color.White);

  // нарисовать игрока
  if(player != null)
    player.Draw(gameTime, spriteBatch);

  // нарисовать матрицу врагов
  enemyGroup.Draw(gameTime, spriteBatch);

  // нарисовать выстрелы игрока
  foreach(PlayerShot playerShot in playerShots)
    playerShot.Draw(gameTime, spriteBatch);

  // вывести набранные очки
#if ZUNE
  spriteBatch.DrawString(arial, "Score", new Vector2(0, 0), Color.White);
  spriteBatch.DrawString(arial, score.ToString(), new Vector2(0, 30),
                         Color.White);
#else
  spriteBatch.DrawString(arial, "Score", new Vector2(50, 50), Color.White);
  spriteBatch.DrawString(arial, score.ToString(), new Vector2(50, 80),
                         Color.White);
#endif
```

```
    // вывести значок с оставшимися жизнями
    livesIcon.Draw(gameTime, spriteBatch);
    spriteBatch.DrawString(arial, "x" + lives.ToString(),
      new Vector2(livesIcon.Position.X + livesIcon.Width + 4,
                  livesIcon.Position.Y + 8),
      Color.White);
  }
```

Теперь при запуске игры вы увидите матрицу врагов, перемещающуюся влево и вправо и постепенно опускающуюся по мере достижения краев экрана.

Дальше нужно, чтобы враги стреляли в игрока, иначе какой интерес играть? Таким образом, необходим класс, противоположный `PlayerShot`, — `EnemyShot`.

Перетащите папку *eshot* в папку *gfx* в конвейере содержимого, чтобы было что рисовать на экране. Эти анимации очень похожи на выстрелы игрока, только в розовом цвете.

Создайте класс `EnemyShot` в папке *Sprites*. Код отличается от класса `PlayerShot` только тем, что в конструкторе загружается другое содержимое. Полная реализация этого класса приведена в примере 1.35.

Пример 1.35. Класс EnemyShot

```
using System.Collections.Generic;
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Content;
using Microsoft.Xna.Framework.Graphics;

namespace AlienAttack
{
  public class EnemyShot : Sprite
  {
    double lastTime;

    public EnemyShot(ContentManager contentManager) : base(contentManager)
    {
      spriteTextures = new Texture2D[2];

      for(int i = 0; i <= 1; i++)
        spriteTextures[i] =
          contentManager.Load<Texture2D>("gfx\\eshot\\eshot_" + i);

      this.Velocity.Y = 3;
    }

    public override void Update(GameTime gameTime)
    {
      if(gameTime.TotalGameTime.TotalMilliseconds - lastTime > 200)
      {
        frameIndex = frameIndex == 0 ? 1 : 0;
        lastTime = gameTime.TotalGameTime.TotalMilliseconds;
      }
```

```
        this.Position += this.Velocity;
      }
    }
  }
```

Следующий шаг – выстрелить в сторону игрока. Для этого проще всего создать генератор случайных чисел и в каждом кадре получать новое значение. Как только полученное значение окажется выше некоего порога, будем создавать новый вражеский выстрел и перемещать его в направлении игрока.

В классе `EnemyGroup` создадим список вражеских выстрелов, присутствующих в данный момент на экране, и настроим генератор случайных чисел (пример 1.36).

Пример 1.36. Конструктор класса EnemyGroup, дополненный подготовкой EnemyShot

```
// все вражеские выстрелы
private List<EnemyShot> enemyShots;

private Random random;

public EnemyGroup(ContentManager contentManager) : base(contentManager)
{
  random = new Random();

  enemyShots = new List<EnemyShot>();

  enemies = new Enemy[EnemyRows, EnemyCols];

  // создаем матрицу врагов
  for(int y = 0; y < EnemyRows; y++)
  {
    for(int x = 0; x < EnemyCols; x++)
    {
      Enemy enemy = new Enemy(contentManager);
      enemy.Position.X = x * enemy.Width + EnemySpacing.X;
      enemy.Position.Y = y * enemy.Height + EnemySpacing.Y;
      enemies[y, x] = enemy;
    }
  }

  enemyWidth = enemies[0, 0].Width;

  // центрируем матрицу по вертикали
  this.Position.X = AlienAttackGame.ScreenWidth/2 -
        ((EnemyCols * (enemyWidth + EnemySpacing.X)) / 2);
  this.Position.Y = EnemyStartPosition;
  this.Velocity.X = EnemyVelocity;
}
```

В методе `Update` вызовем новый метод `EnemyFire`, который получит случайное число и, если оно превышает заданный порог, создаст вражеский выстрел выбранного врага. Метод `NextDouble` объекта `Random` возвращает число в диапазоне от 0 до 1. Если оно больше 0.99, то создаем

новый вражеский выстрел. В методе Update мы также будем обновлять позиции всех выстрелов и удалять из списка те из них, которые оказались за пределами экрана. Код приведен в примере 1.37.

Пример 1.37. Модифицированный метод Update

```
public override void Update(GameTime gameTime)
{
  MoveEnemies(gameTime);
  EnemyFire(gameTime);
}

private void EnemyFire(GameTime gameTime)
{
  // в случайный момент времени создаем вражеский выстрел
  if(random.NextDouble() > 0.99f)
  {
    int x, y;

    // находим еще живого врага
    do
    {
      x = (int)(random.NextDouble() * EnemyCols);
      y = (int)(random.NextDouble() * EnemyRows);
    }
    while(enemies[y,x] == null);

    // создаем сделанный этим врагом выстрел и помещаем его в список
    EnemyShot enemyShot = new EnemyShot(this.Content);
    enemyShot.Position = enemies[y,x].Position;
    enemyShot.Position.Y += enemies[y,x].Height;
    enemyShots.Add(enemyShot);
    AlienAttackGame.AudioManager.PlayCue(AudioManager.Cue.EnemyShot);
  }

  for(int i = 0; i < enemyShots.Count; i++)
  {
    // пересчитать позиции всех выстрелов
    enemyShots[i].Update(gameTime);

    // удалить те, что оказались за пределами экрана
    if(enemyShots[i].Position.Y > AlienAttackGame.ScreenHeight)
      enemyShots.RemoveAt(i);
  }
}
```

Методу Draw осталось пробежаться по списку вражеских выстрелов и нарисовать их на экране. Реализация показана в примере 1.38.

Пример 1.38. Модифицированный метод Draw

```
public override void Draw(GameTime gameTime, SpriteBatch spriteBatch)
{
  // нарисовать всех активных врагов
  foreach(Enemy enemy in enemies)
```

```
    {
      if(enemy != null)
        enemy.Draw(gameTime, spriteBatch);
    }

    // нарисовать все вражеские выстрелы
    foreach(EnemyShot enemyShot in enemyShots)
      enemyShot.Draw(gameTime, spriteBatch);
  }
```

Наконец, добавим метод `Reset`, который очистит экран от объектов `EnemyShot` (пример 1.39).

Пример 1.39. Метод Reset

```
public void Reset()
{
  enemyShots.Clear();
}
```

Сейчас мы имеем почти готовую игру. Игрок может перемещаться и стрелять, враги тоже перемещаются и стреляют. Проблема только в том, что выстрелы проходят сквозь игрока и врагов. Необходимо добавить механизм обнаружения столкновений и взрывы.

Обнаружение столкновений и взрывы

Мы должны знать, когда заряд игрока настигает вражеский корабль, когда выстрелы врагов поражают игрока и когда игрок сталкивается с врагом.

Существует много методов обнаружения столкновений, но мы воспользуемся самым простым, поскольку в нашей игре ничего больше и не требуется: распознаванием пересечения охватывающих прямоугольников.

Идея в том, что вокруг каждого спрайта описан невидимый прямоугольник, касающийся его выступающих краев, как показано на рис. 1.7. Если два таких прямоугольника пересекаются (рис. 1.8), то мы считаем, что объекты коснулись друг друга, и действуем соответственно.

***Рис. 1.7.** Охватывающий прямоугольник, описанный вокруг спрайта*

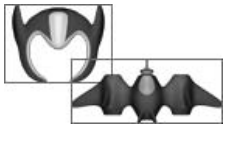

***Рис. 1.8.** Два охватывающих прямоугольника пересекаются*

Реакцией на такое событие является взрыв в месте столкновения. Поэтому, прежде чем писать код обнаружения столкновений, подготовим анимацию взрыва.

Перетащите папку *explosion* из загруженных активов в папку *gfx* конвейера содержимого. Затем создайте новый класс спрайта `Explosion` в папке *Sprites*. Он будет наследовать базовому классу `Sprite` и предоставлять обычные методы загрузки и анимации. Его полный код приведен в примере 1.40.

Пример 1.40. Класс Explosion

```
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Content;
using Microsoft.Xna.Framework.Graphics;

namespace AlienAttack
{
  public class Explosion : Sprite
  {
    double lastTime;

    public Explosion(ContentManager contentManager)
    {
      spriteTextures = new Texture2D[10];

      for(int i = 0; i <= 9; i++)
        spriteTextures[i] =
          contentManager.Load<Texture2D>("gfx\\explosion\\explosion_" + i);
    }

    public new bool Update(GameTime gameTime)
    {
      // если это последний кадр, возвращаем true, чтобы
      // вызывающая программа знала, что мы закончили
      if(frameIndex == 9)
        return true;

      // новый кадр раз в 70 мс
      if(gameTime.TotalGameTime.TotalMilliseconds - lastTime > 70)
      {
        frameIndex++;
        lastTime = gameTime.TotalGameTime.TotalMilliseconds;
      }
      return false;
    }
  }
}
```

Само обнаружение столкновений мы возложим на объект `EnemyGroup`, поскольку он знает, где в каждый момент времени находятся враги и вражеские выстрелы. Начнем со столкновений между выстрелом игрока и врагом. Мы передадим все присутствующие на экране выстрелы игрока новому методу `HandlePlayerShotCollision`. Этот метод обходит

всех находящихся на экране врагов и определяет, пересекаются ли их охватывающие прямоугольники с выстрелами игрока. Если пересечение обнаружено, то в соответствующей позиции выводится анимация взрыва. Код показан в примере 1.41.

Пример 1.41. Метод HandlePlayerShotCollision

```
private List<Explosion> explosions;

public EnemyGroup(ContentManager contentManager) : base(contentManager)
{
  random = new Random();

  enemyShots = new List<EnemyShot>();

  enemies = new Enemy[EnemyRows,EnemyCols];

  explosions = new List<Explosion>();

  // создаем матрицу врагов
  for(int y = 0; y < EnemyRows; y++)
  {
    for(int x = 0; x < EnemyCols; x++)
    {
      Enemy enemy = new Enemy(contentManager);
      enemy.Position.X = x * enemy.Width + EnemySpacing.X;
      enemy.Position.Y = y * enemy.Height + EnemySpacing.Y;
      enemies[y,x] = enemy;
    }
  }

  enemyWidth = enemies[0,0].Width;

  // центрируем матрицу по вертикали
  this.Position.X = AlienAttackGame.ScreenWidth/2 -
        ((EnemyCols * (enemyWidth + EnemySpacing.X)) / 2);
  this.Position.Y = EnemyStartPosition;
  this.Velocity.X = EnemyVelocity;

}

public bool HandlePlayerShotCollision(PlayerShot playerShot)
{
  for(int y = 0; y < EnemyRows; y++)
  {
    for(int x = 0; x < EnemyCols; x++)
    {
      // если выстрел игрока попал во врага, уничтожаем врага
      if(enemies[y,x] != null && CheckCollision(playerShot, enemies[y,x]))
      {
        Explosion explosion = new Explosion(this.Content);
        explosion.Position = enemies[y,x].Position;
        explosions.Add(explosion);
        enemies[y,x] = null;
        return true;
```

```
      }
    }
  }
  return false;
}

public bool CheckCollision(Sprite s1, Sprite s2)
{
  // простое обнаружение столкновений методом
  // охватывающих прямоугольников
  return s1.BoundingBox.Intersects(s2.BoundingBox);
}
```

Обратите внимание, что этот метод (и следующие два) пользуется методом `CheckCollision` из примера 1.41. В нем применяется включенное нами в класс `Sprite` свойство `BoundingBox`, значением которого является библиотечный объект .NET `Rectangle`. Метод `Intersects` объекта `Rectangle` позволяет выяснить, пересекаются ли два прямоугольника.

Далее напишем метод обнаружения столкновений между вражеским выстрелом и игроком – `HandleEnemyShotCollision` (пример 1.42). В нем мы обходим все присутствующие на экране вражеские выстрелы и смотрим, пересекаются ли их охватывающие прямоугольники с прямоугольником, охватывающим корабль игрока.

Пример 1.42. Метод HandleEnemyShotCollision

```
public bool HandleEnemyShotCollision(Player player)
{
  for(int i = 0; i < enemyShots.Count; i++)
  {
    // если вражеский выстрел попал в игрока, уничтожить игрока
    if(CheckCollision(enemyShots[i], player))
    {
      enemyShots.RemoveAt(i);
      return true;
    }
  }
  return false;
}
```

Осталось написать метод `HandleEnemyPlayerCollision`, который проверяет, не пересекается ли какой-нибудь из оставшихся на экране врагов с кораблем игрока (пример 1.43).

Пример 1.43. Метод HandleEnemyPlayerCollision

```
public bool HandleEnemyPlayerCollision(Player player)
{
  for(int y = 0; y < EnemyRows; y++)
  {
    for(int x = 0; x < EnemyCols; x++)
    {
```

```
        // если враг столкнулся с игроком, уничтожить врага
        if(enemies[y,x] != null && CheckCollision(enemies[y,x], player))
        {
          Explosion explosion = new Explosion(this.Content);
          explosion.Position = enemies[y,x].Position;
          explosions.Add(explosion);
          enemies[y,x] = null;
          return true;
        }
      }
    }
    return false;
  }
```

Нам еще нужно обновлять и отображать список взрывов. Модифицируйте методы `Update` и `Draw` класса `EnemyGroup`, как показано в примере 1.44.

Пример 1.44. Модифицированные методы Update и Draw

```
  public override void Update(GameTime gameTime)
  {
    MoveEnemies(gameTime);
    EnemyFire(gameTime);

    for(int i = 0; i < explosions.Count; i++)
    {
      // обновить все взрывы и удалить из списка те,
      // чья анимация завершилась
      if(explosions[i].Update(gameTime))
        explosions.RemoveAt(i);
    }
  }

  public override void Draw(GameTime gameTime, SpriteBatch spriteBatch)
  {
    // нарисовать всех активных врагов
    foreach(Enemy enemy in enemies)
    {
      if(enemy != null)
        enemy.Draw(gameTime, spriteBatch);
    }

    // нарисовать все вражеские выстрелы
    foreach(EnemyShot enemyShot in enemyShots)
      enemyShot.Draw(gameTime, spriteBatch);

    // нарисовать все взрывы
    foreach(Explosion explosion in explosions)
      explosion.Draw(gameTime, spriteBatch);
  }
```

Теперь мы можем включить эти методы в класс `GameScreen` и заняться подсчетом очков. Эта задача возлагается на метод `Update`. Поскольку

этот метод сильно разросся, то я буду объяснять, что в нем делается, по частям. Однако помните, что все эти кусочки представляют собой последовательный код, входящий в один метод `Update`.

Прежде всего, нам понадобится переменная-член для хранения экземпляра анимации взрывающегося корабля игрока. Поместите рядом с объявлениями других членов строку, показанную в примере 1.45.

Пример 1.45. Переменная-член playerExplosion

```
private Explosion playerExplosion;
```

Теперь вернемся к методу `Update` (пример 1.46).

Пример 1.46. Метод Update, часть 1

```
public GameState Update(GameTime gameTime)
{
  MovePlayer(gameTime);
  UpdatePlayerShots(gameTime);

  // если мы еще не проиграли, обновляем позиции врагов
  if(!loseGame)
    enemyGroup.Update(gameTime);

  HandleCollisions(gameTime);

  return GameState.GameScreen;
}

private void HandleCollisions(GameTime gameTime)
{
  // смотрим, попал ли выстрел игрока во врага
  for(int i = 0; i < playerShots.Count; i++)
  {
    PlayerShot playerShot = playerShots[i];
    // проверяем столкновение выстрела с врагом
    if( playerShot != null &&
      enemyGroup.HandlePlayerShotCollision(playerShots[i]))
    {
      // удаляем выстрел, добавляем очки
      playerShots.RemoveAt(i);
      score += 100;
    AlienAttackGame.AudioManager.PlayCue(AudioManager.Cue.Explosion);
    }
  }

  // смотрим, попал ли вражеский выстрел в игрока
  if( player != null &&     enemyGroup.HandleEnemyShotCollision(player))
  {
    // взрываем игрока
    playerExplosion = new Explosion(this.contentManager);
    playerExplosion.Position = player.Position;
    player = null;
    AlienAttackGame.AudioManager.PlayCue(AudioManager.Cue.Explosion);
```

```
    }

    // смотрим, есть ли прямое столкновение игрока с врагом
    if(player != null &&     enemyGroup.HandleEnemyPlayerCollision(player))
    {
      // взрываем игрока
      playerExplosion = new Explosion(this.contentManager);
      playerExplosion.Position = player.Position;
      player = null;
      loseGame = true;
      AlienAttackGame.AudioManager.PlayCue(AudioManager.Cue.Explosion);
    }

    // если анимация взрыва игрока еще не закончилась, обновим ее
    if(playerExplosion != null)
    {
      // если это последний кадр
      if(playerExplosion.Update(gameTime) && !loseGame)
      {
        // удалим анимацию
        playerExplosion = null;

        // если жизней не осталось, мы проиграли
        if(lives == 0)
          loseGame = true;
        else
        {
          // вычесть одну жизнь и привести поле в исходное состояние
          lives--;
          enemyGroup.Reset();
          playerShots.Clear();
          player = new Player(this.contentManager);
        }
      }
    }
  }
```

В выделенном коде вызывается новый метод `HandleCollisions`; он используется процедурами обнаружения столкновений, которые мы поместили в класс `EnemyGroup`, чтобы понять, попал ли в игрока вражеский выстрел или враг с игроком просто столкнулись. В обоих случаях мы создаем новый объект `Explosion` и убираем игрока с экрана. Однако в случае прямого столкновения игра сразу же считается проигранной, и мы устанавливаем переменную `loseGame` в true.

И в последнем фрагменте мы обновляем анимацию взрыва уничтоженного корабля игрока. Если анимация подошла к концу, но игрок еще не проиграл, мы проверяем счетчик жизней. Если жизней не осталось, игра закончена; в противном случае мы уменьшаем счетчик на 1, восстанавливаем начальное положение на экране и создаем новый корабль игрока.

Осталось лишь нарисовать взрыв на экране и вывести сообщение о выигрыше или проигрыше. Окончательная версия метода `Draw` показана в примере 1.47.

Пример 1.47. Окончательный вид метода Draw

```
public void Draw(GameTime gameTime, SpriteBatch spriteBatch)
{
  spriteBatch.Draw(bgScreen, Vector2.Zero, Color.White);

  // нарисовать игрока
  if(player != null)
    player.Draw(gameTime, spriteBatch);

  // нарисовать матрицу врагов
  enemyGroup.Draw(gameTime, spriteBatch);

  // нарисовать выстрелы игрока
  foreach(PlayerShot playerShot in playerShots)
    playerShot.Draw(gameTime, spriteBatch);

  // нарисовать взрыв
  if(playerExplosion != null)
    playerExplosion.Draw(gameTime, spriteBatch);

#if ZUNE
  // вывести набранные очки
  spriteBatch.DrawString(arial, "Score", new Vector2(0, 0), Color.White);
  spriteBatch.DrawString(arial, score.ToString(), new Vector2(0, 20),
    Color.White);

  // вывести значок с оставшимися жизнями
  livesIcon.Draw(gameTime, spriteBatch);
  spriteBatch.DrawString(arial, "x" + lives.ToString(),
  new Vector2(livesIcon.Position.X + livesIcon.Width + 4,
            livesIcon.Position.Y),
    Color.White);
#else
  // вывести набранные очки
  spriteBatch.DrawString(arial, "Score", new Vector2(50, 50), Color.White);
  spriteBatch.DrawString(arial, score.ToString(), new Vector2(50, 80),
    Color.White);

  // вывести значок с оставшимися жизнями
  livesIcon.Draw(gameTime, spriteBatch);
  spriteBatch.DrawString(arial, "x" + lives.ToString(),
  new Vector2(livesIcon.Position.X + livesIcon.Width + 4,
    livesIcon.Position.Y+8),Color.White);
#endif

  // при необходимости вывести подходящий текст
  if(enemyGroup.AllDestroyed())
  {
    Vector2 size = arial.MeasureString("Вы победили!");
    spriteBatch.DrawString(arial, " Вы победили!",
```

```
            new Vector2((AlienAttackGame.ScreenWidth - size.X) / 2,
                        (AlienAttackGame.ScreenHeight - size.Y) / 2),
            Color.Green);
        }

        if(loseGame)
        {
          Vector2 size = arial.MeasureString("Игра закончена");
          spriteBatch.DrawStri ng(arial, "Игра закончена",
            new Vector2((AlienAttackGame.ScreenWidth - size.X) / 2,
                        (AlienAttackGame.ScreenHeight - size.Y) / 2),
            Color.Red);
        }
    }
```

В этой версии рисуется спрайт `playerExplosion`, а затем в центре экрана выводится то или иное сообщение в зависимости от того, удалось ли уничтожить всех врагов или игрок пал их жертвой.

Запуск приложения

Вот теперь программа полностью функциональна! Конечно, это не самая увлекательная в мире игрушка, но неплохая отправная точка для написания собственных игр. Попробуйте!

Поддержка Xbox 360 и Zune

Если у вас есть приставка Xbox 360 и учетная запись в клубе разработчиков (Creator's Club Account) или плеер Microsoft Zune, то вы можете перейти к следующему шагу и запустить игру на любом из этих устройств.

Xbox 360

Создать вариант для Xbox 360 очень просто. Щелкнув правой кнопкой мыши по проекту *AlienAttack* на панели Solution Explorer, вы увидите в контекстном меню пункт Create Copy of Project for Xbox 360 (Создать копию проекта для Xbox 360), как показано на рис. 1.9.

Как следует из названия, эта команда создает в текущем решении новый проект и помещает в него ссылки на существующие файлы с кодом и содержимым. Если собрать этот проект, то получится версия для Xbox 360, которую можно будет установить на вашу приставку.

Если вы еще этого не сделали, то самое время установить на Xbox 360 приложение XNA Game Studio Connect от компании Games Marketplace. Зарегистрировав на сайте Gamertag свою учетную запись в клубе Creators Club Premium, войдите в раздел Games Marketplace и перейдите по ссылке All Games→Browse→All Games→XNA Creators Club. Отсюда вы сможете скачать приложение XNA Game Studio Connect.

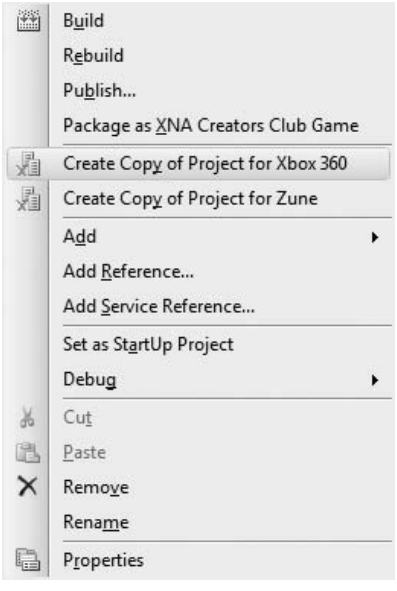

Рис. 1.9. *Пункт меню Create Copy of Project for Xbox 360*

Затем нужно подключить Xbox 360 к ПК. Запустите на Xbox 360 приложение XNA Game Studio Connect, для чего следует выбрать из системного меню команду My Xbox channel→Game Library→Community Games→ XNA Game Studio Connect.

Когда программа запустится, вы увидите окно, содержащее 25-значный ключ подключения (рис. 1.10).

Рис. 1.10. *XNA Game Studio Connect*

Теперь запустите на ПК программу XNA Game Studio Device Center и нажмите кнопку Add Device (Добавить устройство), как показано на рис. 1.11.

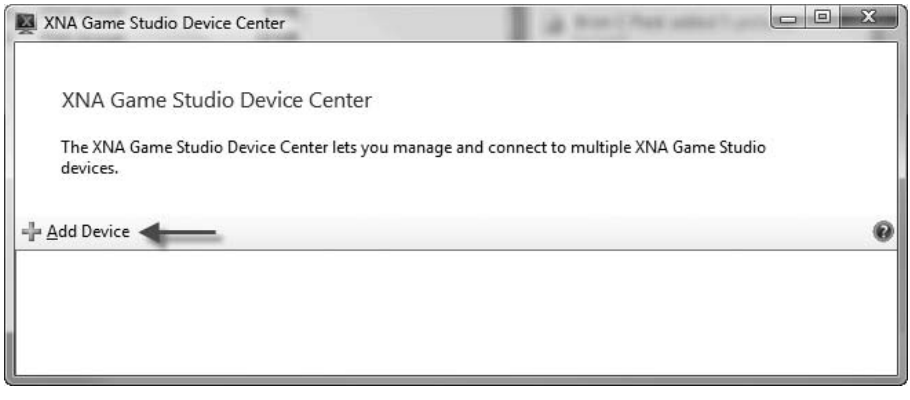

Рис. 1.11. *XNA Game Studio Device Center*

Выберите Xbox 360 в качестве устройства, к которому нужно подключиться (рис. 1.12).

Присвойте Xbox 360 какое-нибудь имя и нажмите кнопку Next (Далее). Введите ключ подключения, отображаемый на экране Xbox 360, и нажмите Next. ПК должен найти приставку Xbox 360 и подключиться к ней. Нажмите кнопку Finish (Готово), чтобы завершить процедуру добавления устройства.

Рис. 1.12. *Выбор Xbox 360*

Теперь, если в Visual Studio вы соберете и запустите проект для Xbox 360, его нужно будет установить на Xbox 360 и запустить. Если вы впоследствии захотите снова запустить игру на Xbox 360, то найдете ее в разделе My Games библиотеки Games Library приставки Xbox 360.

Существует также возможность отлаживать работающую на Xbox 360 игру так, будто она исполняется на ПК. Можно расставлять точки прерывания, проверять значения переменных и т. д.

Zune

Создать проект для Zune чуть сложнее, но все равно нетрудно. Поскольку всю графику необходимо перемасштабировать для экрана 240×320, нам придется пересоздать конвейер содержимого. Для этого проще всего запустить еще один экземпляр Visual Studio и создать новый проект игры для Zune, как показано на рис. 1.13.

Назовите проект *AlienAttackZune* и сохраните его в отдельном каталоге, в стороне от версий для Windows и Xbox 360.

Затем перетащите все загруженные активы из папки Zune в конвейер содержимого. И еще нужно добавить в проект весь написанный код. Создайте такие же папки *Sprites* и *Screens*, как в предыдущем проекте.

В Visual Studio есть возможность помещать в проект ссылки на существующие файлы, а не копировать их целиком. Щелкните правой кнопкой мыши по проекту и выберите из меню команду Add→Existing Item…. Перейдите в корень проекта для Windows, выберите все файлы с расширением *.cs* и нажмите кнопку Add As Link (Добавить как ссылку), как показано на рис. 1.14.

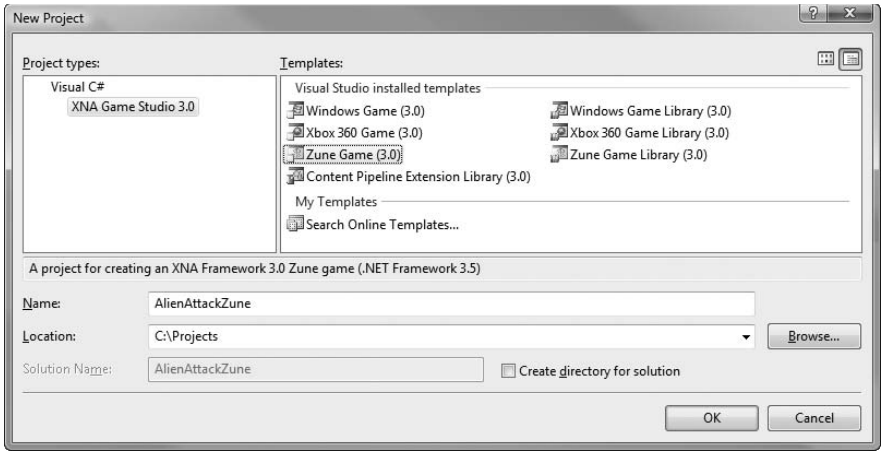

Рис. 1.13. *Создание нового проекта игры для Zune*

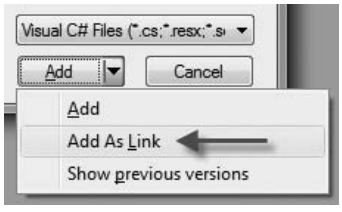

Рис. 1.14. *Кнопка Add As Link*

В результате в новый проект будут добавлены не сами файлы с кодом, а ссылки на них. То же самое проделайте для файлов в папках *Sprites* и *Screens*.

Итак, проект подготовлен, теперь необходимо настроить устройство Zune. Прежде всего, проверьте, что версия прошивки Zune не ниже 3.0. Чтобы узнать номер версии, выберите из меню пункт settings→about→zune. Если версия ниже 3.0, прошивку необходимо обновить.

Если на ваш компьютер не установлено программное обеспечение Zune, можете скачать его с сайта *http://www.zune.net/setup*. Подключите Zune к компьютеру по USB-кабелю, запустите программу Zune и выполните команду Settings→General→Check for Updates (Параметры→Общие→Проверить обновления), как показано на рис. 1.15.

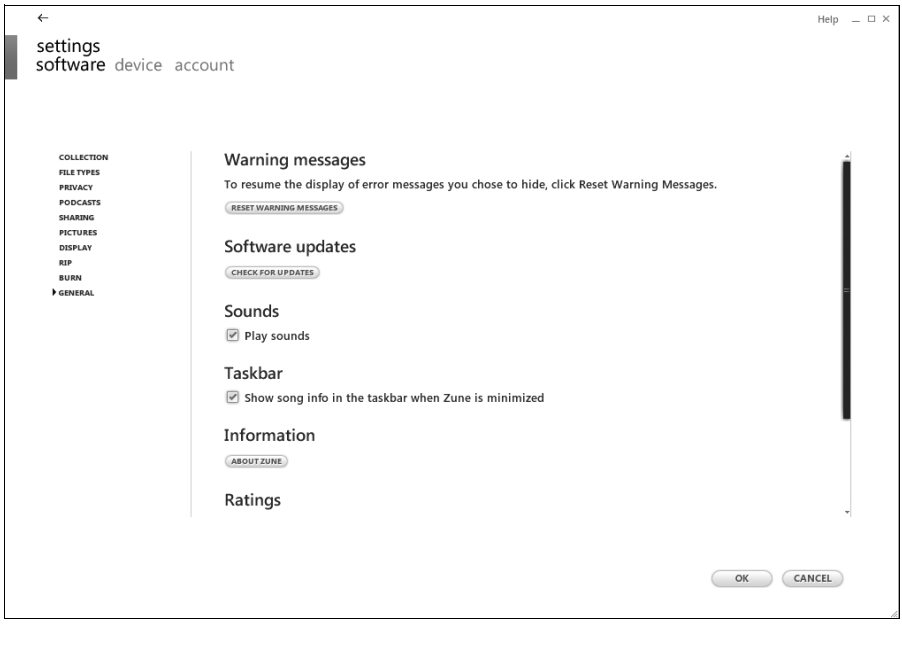

Рис. 1.15. *Проверка наличия обновлений для Zune*

Последняя версия будет скачана и установлена автоматически.

Далее нам предстоит зарегистрировать Zune как устройство для развертывания. Проверьте, что Zune подключен к ПК, запустите XNA Game Studio Device Center и нажмите кнопку Add Device. Выберите в качестве устройства Zune и нажмите Next. Вы увидите список устройств Zune, подключенных к ПК. Выберите, какое вы хотите добавить, и нажмите Next. Устройство будет зарегистрировано.

В Visual Studio запустите проект для Zune; он будет автоматически собран, установлен на устройство Zune и запущен. Как и в случае Xbox 360, версию для Zune можно отлаживать в Visual Studio.

Заключительные замечания

Мы плотно поработали, зато теперь игра «Атака из космоса» работает на Windows, Xbox 360 и Zune, причем отличия между версиями крайне незначительны. Как видите, XNA – весьма мощная библиотека, которая позволяет создавать кросс-платформенные игры без особых усилий. Отталкиваясь от проделанной работы, добавьте новые правила, новых врагов или вообще создайте совсем новую игру!

2

Солдат LEGO: двумерная игра-скроллер на платформе Popfly

Автор	Дэн Фернандес
Сложность	Для начинающих
Необходимое время	6–9 часов
Стоимость	Бесплатно
Программное обеспечение	Silverlight 1.0, LEGO Digital Designer (*http://ldd.lego.com/*) и Paint.NET (*http://www.getpaint.net/*) или эквивалентный графический редактор
Оборудование	Не требуется
Адрес в Интернете	*http://www.popfly.com/users/Dan/LEGOSoldier*

В игре «Солдат LEGO» вы будете управлять специально обученным солдатом, которого послали прекратить незаконные эксперименты по клонированию, проводимые в секретной правительственной лаборатории, расположенной глубоко под землей. Лабораторию возглавляет гнусный негодяй по кличке «Слепой завлаб».

Если вы отважитесь принять на себя эту миссию, то вам предстоит поработать с сайтом (*http://www.popfly.com/*), который позволяет создавать игры, mashup-приложения[1] и веб-страницы прямо в броузере. Для разработки игры «Солдат LEGO» мы воспользуемся инструментом Popfly Game Creator с этого сайта. Он предоставляет интуитивно

[1] Программа, объединяющая данные из нескольких источников. – *Прим. перев.*

понятный, основанный на перетаскивании мышью интерфейс для создания несложных игр с минимальными усилиями.

Краткий обзор

В этой игре вы будете солдатом, который управляет гидравлической машинкой LEGO, позволяющей крушить различных врагов: от вооруженных автоматами охранников до безмозглых зомби (рис. 12.1).

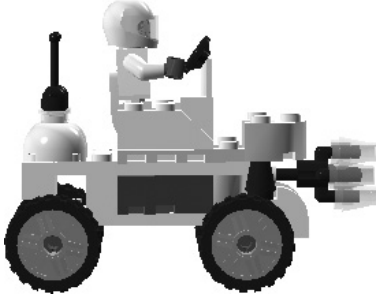

Рис. 2.1. *Главный персонаж, которым вы управляете*

Блуждая в подземельях, вы научитесь пользоваться различными предметами, например, катапультой для перепрыгивания через препятствия или пороховыми бочками для взрывания врагов (рис. 2.2).

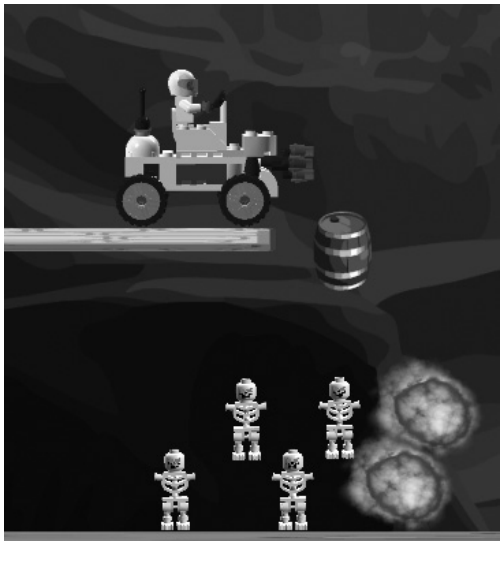

Рис. 2.2. *Взрывание скелетов с помощью пороховых бочек*

Свое творение вы сможете опубликовать на сайте Facebook, чтобы ваши друзья могли посостязаться в расстраивании планов Слепого завлаба (рис. 2.3).

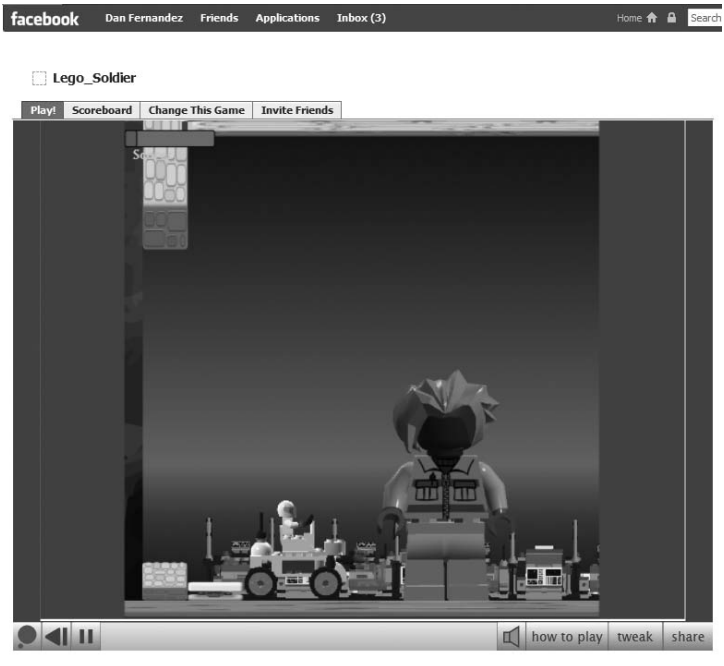

Рис. 2.3. *Публикация игр на сайте Facebook*

Терминология Popfly Game Creator

Прежде чем приступать к делу, познакомимся с некоторыми терминами Popfly Game Creator, которые нужно знать для создания игры.

Сцена

Это место, где размещается пользовательский интерфейс игры. Для Солдата LEGO мы создадим одну длинную сцену.

Область просмотра

Та часть сцены, которая видна в данный момент. В Солдате LEGO область просмотра включается, чтобы выводилась не вся сцена разом, а только область просмотра.

Персонаж

Графический элемент на сцене. Технически на сцену можно помещать элементы Silverlight, не являющиеся персонажами, например

элемент управления TextBlock, но персонажи отличаются тем, что обладают поведением. К числу персонажей в Солдате LEGO относятся фоновая картинка сцены, наше главное действующее лицо, стены, враги и все остальное.

Поведения

Это логика работы персонажа или сцены. Например, можно определить двигательное поведение, в соответствии с которым персонаж будет перемещаться.

Свойства

На панели свойств вы задаете атрибуты персонажа или сцены. Например, можно установить прозрачность персонажа в диапазоне от 0 до 1, где 0 означает, что персонаж полностью прозрачен, то есть невидим, а 1 – что он абсолютно непрозрачен.

Игровое поле

Здесь вы конструируете свойства для своей игры, отсюда же загружаете разнообразные активы: изображения, звуковое сопровождение и видео.

Конструирование солдата LEGO: шаг за шагом

Определив основные термины, рассмотрим, какие шаги предстоит предпринять для создания игры Солдат LEGO. Вот они:

1. Создание Popfly-игры.
2. Конструирование персонажей.
3. Конструирование сцены.
4. Создание поведений сцены.
5. Наделение персонажей «разумом» с помощью поведений.
6. Публикация игры на сайте Facebook.

Создание Popfly-игры

Первым делом необходимо создать игру. Для этого зайдите на сайт *www.popfly.com* и нажмите кнопку «Create a Game». Вас попросят либо ввести идентификатор существующего пользователя Popfly, либо создать новую учетную запись, если вы еще не регистрировались.

После регистрации вы попадете в конструктор Popfly-игр, где вас приветствует мастер создания игры, показанный на рис. 2.4. Выберите вариант «...or start from scratch» (начать с нуля).

В данный момент мы создали пустую заготовку игры. Сохраните ее, нажав кнопку Save в правом верхнем углу окна, и назовите свой проект, например, *LEGOSoldier*.

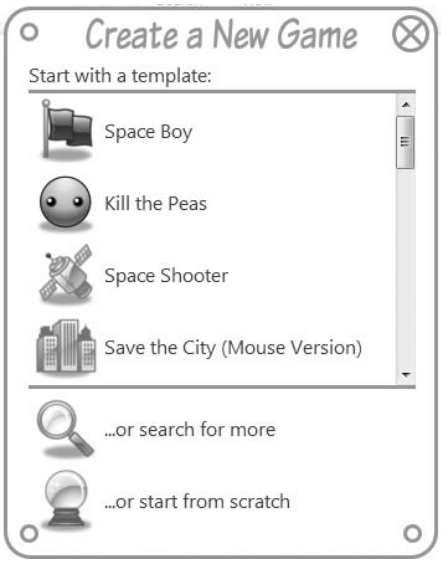

Рис. 2.4. *Меню создания новой игры*

Проектирование персонажей

На сайте Popfly есть сотни готовых персонажей, но, коль скоро мы разрабатываем игру LEGO, то, очевидно, нужны персонажи в стиле игрушек этой фирмы. Popfly позволяет всем желающим создавать новых персонажей и помещать их в репозиторий; так мы и поступим со своими моделями LEGO. У вас, возможно, блестящие способности к графическому дизайну, но есть люди, которым не так повезло, поэтому мы воспользуемся бесплатной программой Digital Designer от компании LEGO для построения 3D-моделей в среде, где основным инструментом является буксировка мышью. Вокруг Digital Designer тоже сформировалось обширное сообщество почитателей, которые загружают свои модели LEGO, передавая их во всеобщее пользование. Для Солдата LEGO мы скачаем и модифицируем несколько созданных другими пользователями моделей: башню, лабораторию и другие.

Для создания своих персонажей нужно сделать следующее:

1. Сконструировать модель LEGO с помощью программы LEGO Digital Designer.
2. Экспортировать один или несколько снимков модели LEGO, которую мы собираемся использовать.
3. Убрать из картинки лишнее пустое место.
4. Загрузить получившееся изображение в Popfly.

5. Сконструировать несложную XAML-разметку с описанием изображения.
6. Создать в Popfly новый персонаж.

Конструирование модели с помощью LEGO Digital Designer

Для создания модели главного персонажа (Main) откройте LEGO Digital Designer, щелкните по картинке Choose Free Build (Выберите бесплатное создание) и раскройте панель «деталей» Brick Palette, на которой находятся готовые детали LEGO. Теперь можно создавать модели LEGO, перетаскивая детали на поверхность конструирования. На рис. 2.5 показан окончательный результат конструирования главного персонажа Солдата LEGO. Мы просто собрали его из готовых деталей, соединив их между собой в конструкторе.

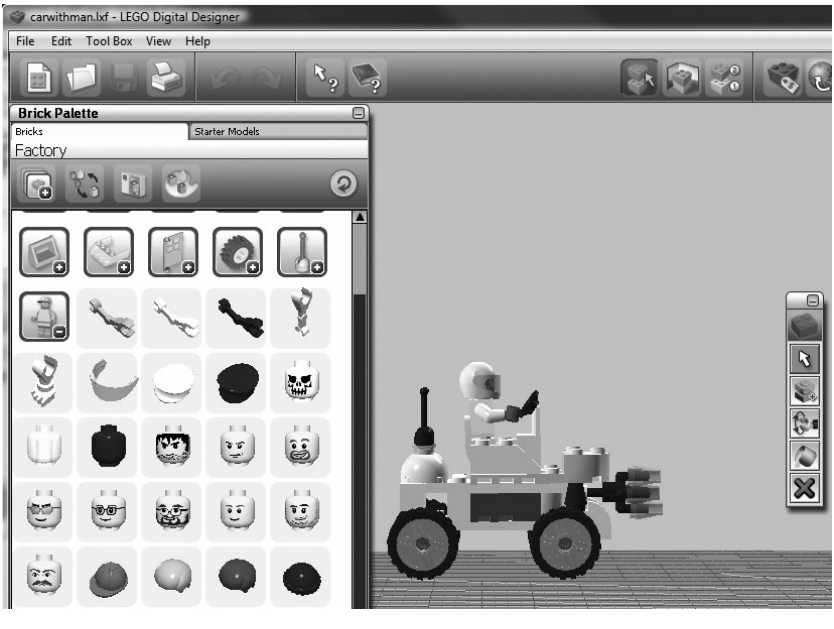

Рис. 2.5. *Создание своей модели в программе LEGO Digital Designer*

Экспорт модели LEGO

Закончив конструирование модели главного персонажа в Digital Designer, поверните ее, чтобы она выглядела плоской, и выполните команду Toolbox→Take a Screenshot (Инструменты→Сделать снимок). В результате будет экспортирован снимок модели точно в том виде, в каком вы ее видите

на экране. Digital Designer хранит снимки в формате Portable Network Graphics (PNG) с поддержкой прозрачности. Все снимки сохраняются в папке *Мои рисунки* текущего пользователя и именуются *LDDScreenshot<x>.png*, где *<x>* – последовательные целые числа (1, 2, 3...).

Обрезка пустого места

Созданный программой Digital Designer снимок имеет те же размеры, что окно приложения. Так, если размер окна равен 1280×1024, то и снимок получится такого же размера. Понятно, что при этом образуется много пустого (прозрачного) места, которое нам совершенно ни к чему, поэтому нужно взять какой-нибудь графический редактор и удалить все лишнее.

Для этой цели мы воспользуемся приложением Paint.NET – бесплатным и удобным редактором, который можно скачать с сайта *http://www.getpaint.net/*. Откройте снимок в Paint.NET и с помощью инструмента Rectangle Select (Прямоугольное выделение) выделите только саму модель LEGO. После этого выберите из меню команду Crop to Selection (Обрезать), как показано на рис. 2.6. В результате все лишнее пустое

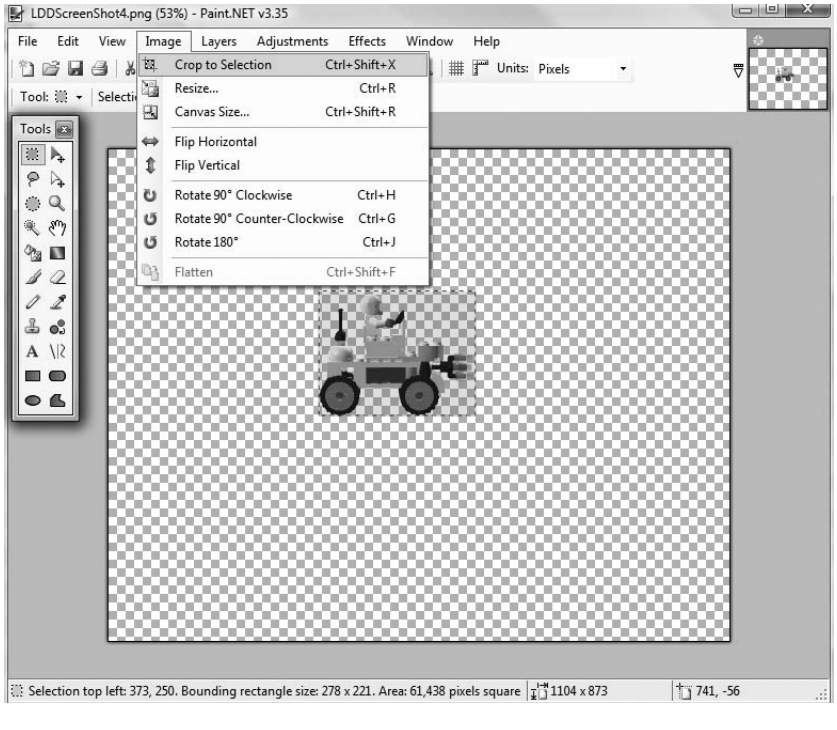

***Рис. 2.6.** Удаление лишнего пустого места со снимка модели в программе Paint.NET*

место будет убрано, изображение станет меньше и с ним станет проще работать. Не забудьте сохранить обрезанное изображение и попутно присвойте ему более осмысленное имя, например *MainCharacter.png*.

Загрузка изображений на сайт Popfly

Итак, изображение готово, давайте загрузим его на сайт Popfly. Для этого вернитесь в созданную ранее Popfly-игру, выберите из меню пункт Game и под текстом Add a file from… (Добавить файл из) отметьте переключатель your computer (ваш компьютер), чтобы загрузить изображение со своего компьютера. Нажмите кнопку Browse… (Обзор) и найдите только что созданный файл *MainCharacter.png* (рис. 2.7).

Рис. 2.7. *Загрузка изображений на сайт Popfly из меню Game*

Загруженное изображение будет иметь такой URL:

http://www.popfly.com/users/<UserName>/<ProjectName>/<ImageName>.png

Так, для пользователя по имени Dan загруженный файл *MainCharacter.png* имеет адрес

http://www.popfly.com/users/Dan/LegoSoldier/MainCharacter.png

Для адресации своего файла не нужно указывать его полный URL, а следует воспользоваться макросом `$base$`. С помощью этого макроса

мы можем сослаться на изображение внутри Popfly по имени *$base$/MainCharacter.png*, которое будет автоматически транслировано в полный URL.

Почему следует пользоваться макросом $base$

Макрос `$base$` является предпочтительным способом ссылки на игровые активы (изображения, звуковые эффекты и т. д.) из-за функции `Rip`. Эта функция позволяет одному пользователю Popfly создавать персональные копии проекта другого пользователя.

Пусть есть два пользователя Popfly: Энджи и Дэн. Энджи создала и сделала всеобщим достоянием Popfly-игру, а Дэн ее игру скопировал (ripped). Если бы Энджи включила полные пути, например *www.popfly.com/users/Angie/MyProject/img1.png*, то в скопированном проекте остались бы «зашиты» ссылки на файлы Энджи (например, *Angie/MyProject/img1.png*). Это плохо, так как если Энджи впоследствии удалит файл *img1* или заменит его чем-то совершенно другим, то игра Дэна перестанет работать.

Но, если вместо полных путей Энджи будет ссылаться на свое изображение с помощью макроса `$base$` (*$base$/img1.png*), то в процессе копирования ее проекта Popfly создаст локальные копии файлов для Дэна, например *Dan/RippedProject/img1.png*. Теперь Энджи может делать со своим файлом *img1.png* что угодно – на копии проекта, сделанной Дэном, это никак не отразится.

Создание персонажа в Actor Designer

Создав все нужные изображения, вернитесь в Popfly и щелкните по значку Actors, чтобы перейти в конструктор персонажей (Actor Designer). Нажмите кнопку New для создания нового персонажа (рис. 2.8).

Рис. 2.8. *Пункт меню New*

Приведем краткий обзор элементов конструктора персонажей, показанных на рис. 2.9.

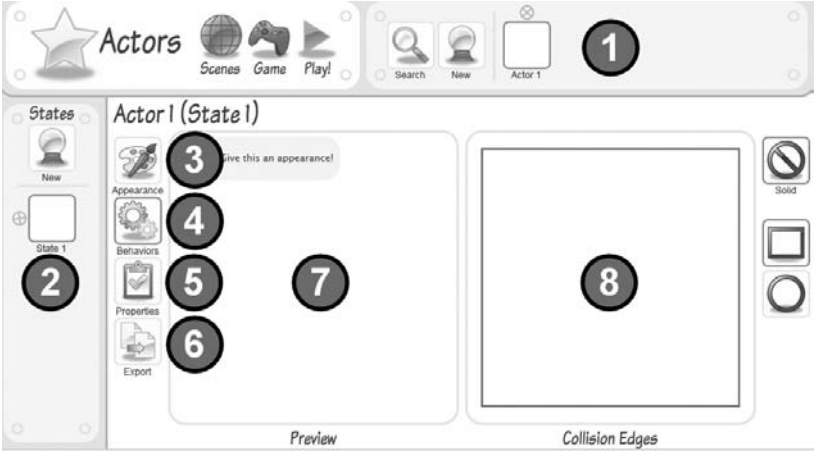

Рис. 2.9. *Конструктор персонажей в Popfly*

1. Список персонажей: это список определенных в вашей игре персонажей.
2. Состояния (States): здесь можно просмотреть имеющиеся состояния персонажа и создать новое. Состояния располагаются одно под другим.
3. Внешний вид (Appearance): здесь определяется внешний вид текущего состояния.
4. Поведения (Behaviors): это управляющая логика, которая стоит за действиями персонажа.
5. Свойства (Properties): это механизм сохранения переменных игры или персонажа, например текущего счета или здоровья игрока.
6. Экспорт (Export): позволяет поделиться созданными вами персонажами с сообществом.
7. Предварительный просмотр (Preview): показывает текущий внешний вид персонажа.
8. Края (Collision Edges): используется для распознавания столкновений персонажей. Вы можете сами определить края, буксируя красную рамку. Можно также отключить опцию Solid (Твердый), тогда персонаж не будет возбуждать события столкновения. Это удобно для персонажей на заднике сцены, например облаков или города на горизонте, с которыми ваш главный герой не должен сталкиваться.

Чтобы задать внешний вид главного персонажа, нажмите кнопку Appearance (#3 на рис. 2.9), а затем кнопку Switch to XAML (Перейти в режим XAML). В реальности персонаж представляет собой код на языке разметки Silverlight XAML. Вам необходимо определить высоту, ширину и изображение персонажа, как показано в примере 2.1.

Пример 2.1. Простая XAML-разметка элемента управления Image

```
<!-- XAML персонажа Main -->
<Canvas xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
   Width="419" Height="336">
<Image Width="419" Height="336" Source="$base$/MainCharacter.png" />
</Canvas>
```

Скопируйте этот код в поле для задания XAML, как показано на рис. 2.10.

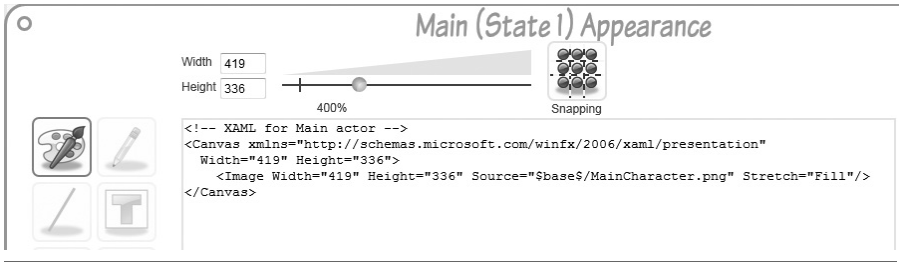

Рис. 2.10. *Задание XAML-разметки, описывающей внешний вид персонажа Main*

Сохраните XAML-разметку и покиньте это окно. Напоследок назовем наш персонаж как-то более осмысленно. Щелкните по имени персонажа и измените «Actor1» на «Main».

Когда все будет сделано, персонаж Main должен выглядеть, как показано на рис. 2.11.

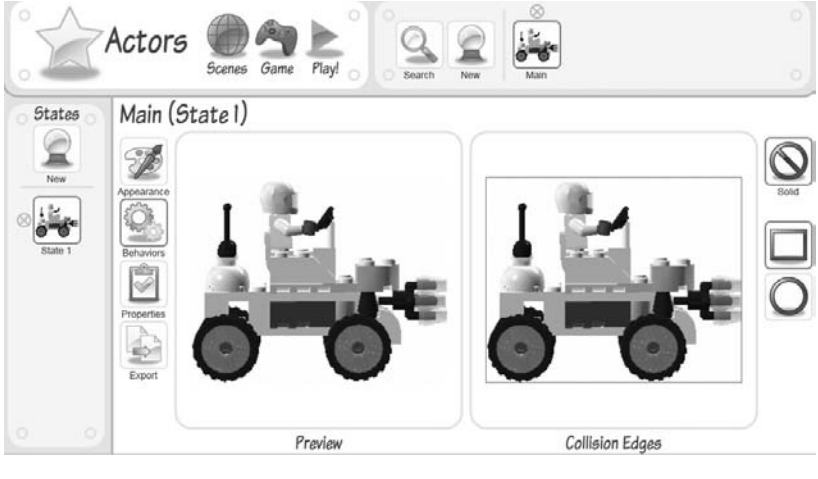

Рис. 2.11. *Окончательный вид персонажа Main*

Добавление индикатора здоровья и текущего счета

Напомним, что в игре любой визуальный элемент – это персонаж, поэтому необходимо создать персонаж для индикатора здоровья и текущего счета. Индикатор здоровья есть во многих играх, выглядит он как полоска переменного размера. Игрок, которому был нанесен ущерб, сразу не погибает, но размер полоски уменьшается.

Создайте нового персонажа, как и раньше, и назовите его `HealthBar`. Затем нажмите кнопку Switch to XAML для перехода в режим XAML и скопируйте в элемент управления TextBlock код из примера 2.2. После возврата в режим конструирования (Design view) появится внешнее представление нового персонажа (рис. 2.12).

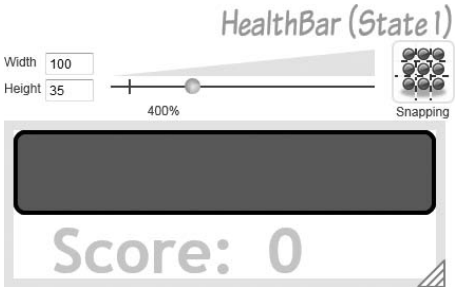

Рис. 2.12. *Внешний вид персонажа, описывающего индикатор здоровья и текущий счет*

Пример 2.2. XAML-разметка индикатора здоровья и текущего счета

```
<Canvas xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
  Width="100" Height="35" x:Name="HealthBar">
  <Rectangle Width="100" Height="20" Fill="Gray" Stroke="Black"
    RadiusX="2" RadiusY="2" Canvas.ZIndex="1"/>
  <Rectangle Width="100" Height="20" Name="HealthValue" Fill="Green"
    Stroke="Black" RadiusX="2" RadiusY="2" Canvas.ZIndex="2"/>
  <TextBlock Width="50" Height="20" Text="Score:" FontFamily="Trebuchet MS"
    FontWeight="Bold" Foreground="LightBlue" Canvas.Left="9" Canvas.Top="20"/>
  <TextBlock Width="30" Height="20" Name="ScoreValue" Text="0"
    Foreground="LightBlue" FontFamily="Trebuchet MS" FontWeight="Bold"
    Canvas.Top="20" Canvas.Left="60" TextWrapping="Wrap"/>
</Canvas>
```

Сделаем несколько замечаний по поводу кода в примере 2.2. В элементе управления `Canvas` мы определяем прямоугольники `Rectangle` серого и зеленого цвета, расположенные один поверх другого. Когда персонаж Main получит повреждение, мы будем уменьшать ширину зеленого прямоугольника, открывая находящийся под ним серый (рис. 2.13). Определены также два текстовых блока `TextBlock`: один содержит над-

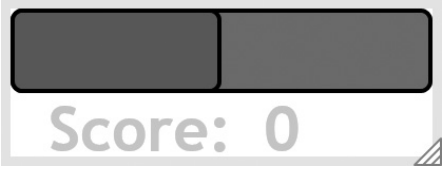

Рис. 2.13. *Когда ширина индикатора здоровья уменьшается, становится виден находящийся под ним серый прямоугольник*

пись Score (Очки), другой – собственно набранные очки. Как можно динамически обновлять индикатор здоровья, мы расскажем ниже в разделе «Создание поведений индикатора здоровья».

Импорт персонажей

Чтобы не тратить время и силы, вы можете пропустить процедуру создания и импорта дополнительных персонажей LEGO и просто импортировать уже готовых персонажей, снабженных меткой «LEGO Soldier». Все они были созданы точно так же, как описано выше.

Чтобы добавить персонажей, нажмите кнопку Search, введите строку «LEGO Soldier» и щелкните по каждому из пяти найденных персонажей (рис. 2.14).

Помимо персонажей LEGO мы хотим добавить еще некоторых встроенных в Popfly персонажей для фона и других особенностей игры. Сами встроенные персонажи и их имена изображены на рис. 2.15. Чтобы добавить нужного, найдите его по имени и щелкните по нему. Если смотреть слева направо, то персонажи называются Stone Block (Каменная стенка), Cave (Пещера), Dark Room (Темная комната), Dirt Block (Земляная стенка), Rock-Horizontal (Лежащий камень), Sign – Up (Указатель вверх), Sign – Right (Указатель вправо), Barrel (Бочка) и Catapult (Катапульта).

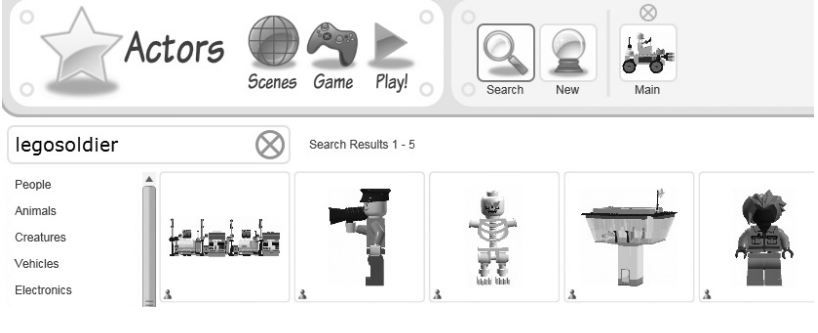

Рис. 2.14. *Добавление готовых персонажей LEGO*

Рис. 2.15. *Импорт встроенных в Popfly персонажей*

Собрав всех персонажей, мы можем перейти в конструктор сцены и расположить на ней персонажей.

Конструирование главной сцены

Итак, персонажи готовы, давайте соберем из них сцену, на которой будет разворачиваться игра. Мы уже говорили, что сцена – это место, где конструируется пользовательский интерфейс игры. Познакомимся сначала с компонентами конструктора сцены, который изображен на рис. 2.16.

1. Список сцен: по умолчанию для каждой новой Popfly-игры создаются следующие сцены:
   a. Вступление (Intro): заставка перед началом игры.
   b. Главная (Main): первая сцена, загружаемая вслед за заставкой.
   c. Выигрыш (Won): загружается, когда вы выиграли.
   d. Проигрыш (Lost): загружается, когда вы проиграли.
   e. Правила (How to Play): сцена с объяснением правил игры.
2. Список персонажей: персонажи, включенные в игру. Из этого списка можно перетащить персонажа на поверхность конструирования.
3. Размер сцены: здесь задается ширина, высота и масштаб сцены. Для масштабирования протяженных сцен пользуйтесь предоставляемым инструментом.
4. Привязка: упрощает выравнивание персонажей по горизонтали и вертикали. Этот режим можно отключить, если требуется располагать персонажей в произвольных местах сцены.
5. Область просмотра (Viewport): позволяет ограничить часть сцены, видимую в любой момент времени. Представлена синим квадратиком в конструкторе сцены.
6. Поверхность конструирования: область экрана, на которую вы перетаскиваете персонажей.
7. Список персонажей на сцене: в отличие от собственно списка персонажей, в этом списке присутствуют экземпляры персонажей, находящихся на текущей сцене. Если на сцене находится пять скелетов, то каждый будет представлен в этом списке отдельно.

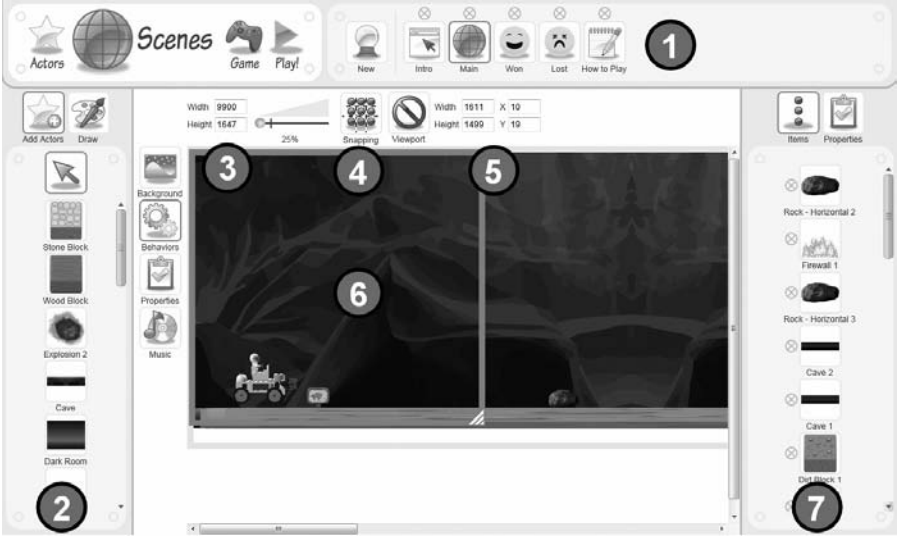

Рис. 2.16. *Конструктор сцены в Popfly Game Creator*

Конструирование главной сцены

Главной сценой (Main) в нашей игре будет просматриваемая слева направо область, описывающая многоуровневую пещеру. По ходу дела мы добавим LEGO-персонажей: башню, зомби, пороховые бочки и других. Все они встретятся на пути к Слепому завлабу. На рис. 2.17 показана главная сцена, уменьшенная так, чтобы ее можно было увидеть целиком.

Прикинув, как должна выглядеть полная сцена, приступим к ее пошаговому конструированию. Первым делом нажмите кнопку Viewport (#5 на рис. 2.16), чтобы включить для сцены режим области просмотра. Затем задайте размеры сцены:

Размеры сцены

Ширина: 9900, высота: 1650

Рис. 2.17. *Полный вид главной сцены*

Размеры области просмотра

Ширина: 1600, высота: 1650, X: 0, Y: 0

В результате будет создана протяженная по горизонтали сцена и область просмотра, позволяющая в каждый момент времени видеть только часть сцены. Значения X и Y определяют положение начальной области просмотра; так как сцена скользит слева направо, то первоначально область просмотра находится в точке 0,0.

Добавление двух персонажей фона пещеры

Теперь необходимо добавить двух персонажей, представляющих собой плитки, которыми мостится фоновая картинка игры. Таких плиток будет две. Перетащите их на поверхность конструирования из списка персонажей (#2 на рис. 2.16).

Примечание

Если поместить персонажа в нужное место не получается, попробуйте отключить режим привязки (#4 на рис. 2.16). Это даст возможность размещать персонажей в произвольных точках сцены. Кроме того, для перемещения по такой большой сцене, как эта, очень пригодится функция масштабирования (#3 на рис. 2.16).

После размещения пещеры (Cave) появляются синие размерные указатели, показывающие ширину и высоту персонажа. Поместите этот элемент фона в левый верхний угол (0,0) и растяните до ширины 4950 и высоты 1650.

На рис. 2.18 показана уменьшенная поверхность конструирования после изменения размеров персонажа Cave.

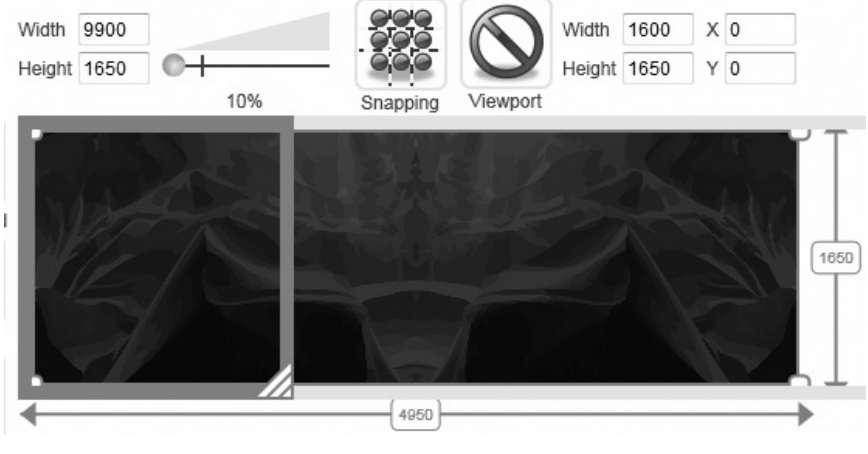

Рис. 2.18. *Изменение размера персонажей на поверхности конструирования*

Мощение фона

Поместив персонажа туда, куда нужно, вы можете замостить им область сцены, то есть скопировать его с такими же размерами и поместить справа вплотную к его предыдущему изображению. Для этого выберите персонаж Cave и воспользуйтесь механизмом копирования и вставки (Ctrl-C/Ctrl-V). Копия будет автоматически выровнена и расположена точно справа от исходного персонажа (рис. 2.19).

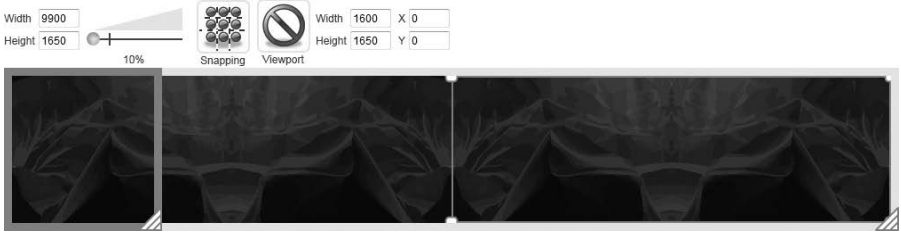

Рис. 2.19. *Для автоматического мощения пользуйтесь копированием и вставкой*

Мощение пола

Итак, фон готов. Теперь сконструируем пол, перетащив персонажа Dirt Block (Земляная стенка) и задав его ширину (4950) и высоту (55). Поместите плитку Dirt Block в левый нижний угол сцены, как показано на рис. 2.20. С помощью клавиш Ctrl-C и Ctrl-V поместите копию Dirt Block справа от оригинала. Теперь вдоль нижнего края обеих плиток Cave расположен пол, составленный из плиток Dirt Block.

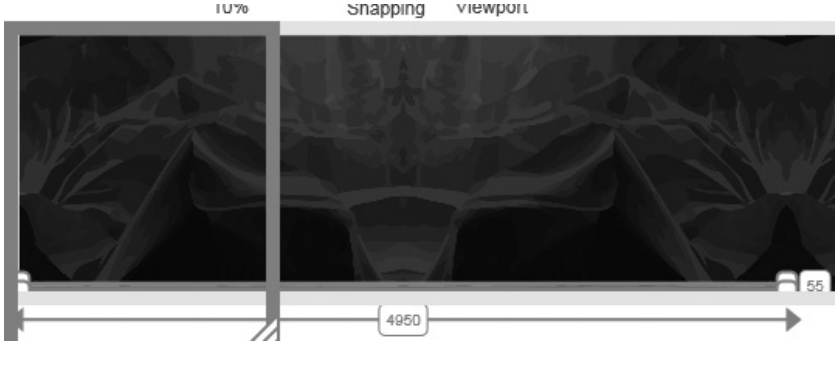

Рис. 2.20. *Персонаж Dirt Block располагается вдоль нижнего края сцены*

Конструирование первой половины сцены

В первой половине сцены будут находиться персонаж Main, кое-какие враги и персонажи, составляющие фон и антураж (рис. 2.21). Мы будем описывать шаги по их добавлению последовательно, но при конструировании сцены имейте в виду, что точные позиции персонажей не существенны. Не бойтесь экспериментировать с фоном, персонажами и позиционированием.

Рис. 2.21. *Персонажи в первой половине сцены*

Как показано на рис. 2.21, мы добавили ряд персонажей, пометив их номерами:

1. Индикатор здоровья (HealthBar). Отображает текущее здоровье главного персонажа и количество набранных очков. Задайте для этого персонажа ширину 300 и высоту 100.
2. Главный персонаж (Main). Это наш главный герой. Изменять его размеры не нужно.
3. Стрелка вправо. Визуальное напоминание новым игрокам о том, что они должны продолжать движение вправо. Задайте для него ширину и высоту 100.
4. Охранник LEGO. Это первый оказавшийся на пути враг, который может стрелять в героя. Изменять его размеры не нужно.
5. Башня LEGO. Выступает в роли сторожевой вышки. Задайте для нее ширину 600 и высоту 700.
6. Три скелета LEGO. Это медлительные враги. Изменять их размеры не нужно.
7. Лежачий камень. Герой должен через него перепрыгнуть. Задайте для него ширину и высоту 130.
8. Стрелка вверх. Визуальное напоминание игроку, впервые столкнувшемуся с катапультой, о том, что нужно подпрыгнуть вверх. Задайте ширину и высоту 100.
9. Катапульта. Если герой запрыгнет на катапульту, то она подбросит его в воздух. Задайте для этого персонажа ширину 270 и высоту 50.

Конструирование второй половины сцены

На второй половине сцены есть несколько уровней, пороховые бочки и лаборатория Слепого завлаба (рис. 2.22).

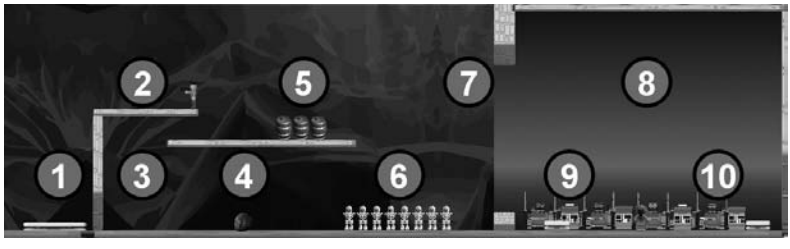

Рис. 2.22. *Персонажи во второй половине сцены*

На вторую половину сцены мы тоже поместим ряд персонажей (рис. 2.22):

1. Катапульта. Это та же катапульта, что была в первой половине сцены. Мы поместили ее сюда, чтобы не прерывалась визуальная связь между обеими половинами.
2. Горизонтальная каменная стенка и охранник. Это приподнятая платформа, на которую можно попасть только с помощью катапульты. На платформе находится охранник. Задайте размер стенки 750×50, а размер охранника не изменяйте.
3. Вертикальная каменная стенка, не дающая герою двигаться вправо. Герой вынужден воспользоваться катапультой, чтобы запрыгнуть на платформу. Задайте ширину 50 и высоту 830.
4. Лежачий камень. Такой же, как в первой половине сцены; герой должен через него перепрыгнуть. Поскольку этот камень ничем не отличается от встречавшегося ранее (#7 на рис. 2.21), можете скопировать последний с помощью Ctrl-C/Ctrl-V, при этом размеры будут сохранены.
5. Каменная стенка и бочки. Эта стенка помещена примерно посередине сцены по вертикали и имеет размеры 1350×50. Три персонажа Barrel представляют пороховые бочки. Если герой столкнет их вниз, то они упадут на скелетов и подорвут их. Задайте для одной бочки ширину 120 и высоту 165, а затем дважды скопируйте ее, чтобы на платформе стояли три одинаковых бочки.
6. Восемь скелетов LEGO. Поместите под платформой несколько скелетов (#5), чтобы скинутые бочки приземлились точно на врагов.
7. Две каменные стенки. Эти персонажи образуют что-то вроде прохода, отделяющего пещеру от лаборатории. Для верхней стенки задайте размер 150×450, а для нижней – 150×125.

8. Темная комната, каменные стенки сверху и справа. Это лаборатория, где герою предстоит сразиться с главным злодеем – Слепым завлабом. Чтобы лаборатория внешне отличалась от пещеры, мы воспользовались персонажем Dark Room, занимающим все место за дверью (#7). Задайте для этого персонажа ширину 2050 и высоту 1625. Кроме того, добавьте потолок и правую стену, взяв для этого две каменные стенки. Размер персонажа, обозначающего потолок, пусть будет 1950×50, а обозначающего правую стену – 75×1650.
9. Две лаборатории LEGO и катапульта. В этой игре ни застрелить, ни задавить Слепого завлаба не получится (при первом столкновении он вырастает в четыре раза). Зато можно – и нужно – прыгнуть ему на голову, воспользовавшись находящейся в лаборатории катапультой. Чтобы придать помещению вид лаборатории, мы взяли две лаборатории LEGO и расположили их рядышком. Задайте для одной из них ширину 780 и высоту 280, после чего скопируйте и поместите копию справа. Поскольку Слепой завлаб будет увеличиваться в размерах, нам необходимы маленькие катапульты, с помощью которых герой сможет подпрыгнуть и опуститься на макушку завлаба. Поместите катапульту слева от лаборатории LEGO и задайте для нее размер 185×60.
10. Слепой завлаб и катапульта. Поместите Слепого завлаба в центр лаборатории, не изменяя его размеров, а в дальнем конце лаборатории (справа) расположите еще одну катапульту. Размеры последней сделайте такими же, как для катапульты #9.

Вот мы и сконструировали всю главную сцену, которая должна выглядеть, как показано на рис. 2.17. Следующий шаг – добавить свойства, в которых будет храниться состояние здоровья главного персонажа и счет игры.

Создание свойств игры

Нам необходимо хранить количество набранных очков и текущее здоровье персонажа Main в свойствах игры. Поскольку свойство Score (Счет) создается автоматически вместе с игрой, то нам нужно добавить только свойство Health (Здоровье). Для этого перейдите на вкладку Game и щелкните по ссылке Add a property (Добавить свойство). Назовите новое свойство Health и присвойте ему начальное значение 100, как показано на рис. 2.23. Необходимо также нажать кнопку higher score is better (чем счет выше, тем лучше), чтобы встроенный в Popfly механизм подсчета рекордов знал, как ранжировать результаты.

Создание поведений

Определив свойства игры, мы теперь должны наделить персонажей поведением. Поведение – это «мозги» персонажа, то есть правила, опи-

Рис. 2.23. *Добавление свойства Health, необходимого для индикатора здоровья*

сывающие, что персонаж может и чего не может делать: перемещаться, бегать, прыгать и т. д. Бывают и поведения в ответ на события, например увеличение количества набранных очков после уничтожения монстра.

На рис. 2.24 показано, какими типами поведений можно наделить персонажа.

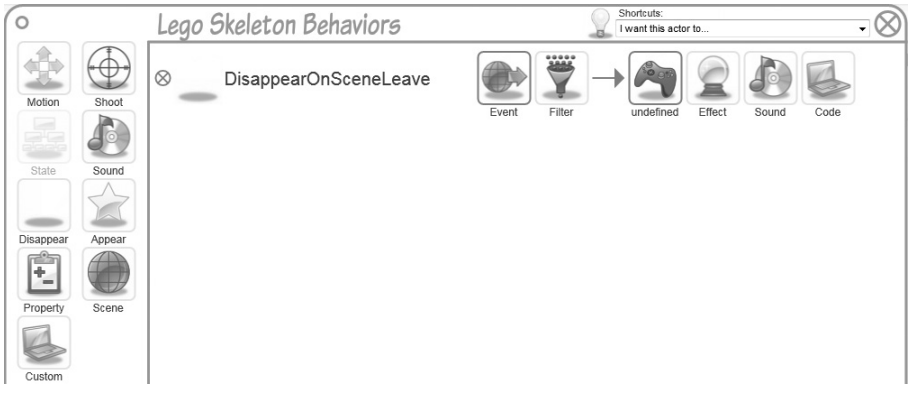

Рис. 2.24. *Конструктор поведений персонажа*

1. Ярлыки (Shortcut): это подобия макросов, позволяющие добавить несколько поведений для типичных задач. Например, вместо того чтобы определять отдельные поведения для перемещения влево/вправо и прыжков, можно добавить поведение, описывающее бег и прыжки, щелкнув по ярлыку run and jump by pressing keys (бегать и прыгать по нажатию клавиш).
2. Список поведений: новые поведения добавляются в этот список.
3. Перемещение (Motion): позволяет определить движение персонажа. Мы неоднократно будем задавать это поведение для наших персонажей.
4. Стрельба (Shoot): позволяет стрелять, например, пулями. Пригодится для вооруженных персонажей.
5. Состояние (State): это поведение позволяет изменять состояние персонажа. Полезно для персонажей с двумя состояниями, например «смотрит влево» и «смотрит вправо», которые пользователь переключает нажатием клавиш со стрелкой влево или вправо.
6. Звук (Sound): позволяет воспроизводить звук по некоторому событию.
7. Исчезновение (Disappear): позволяет управлять видимостью персонажа. Например, если пуля попала во врага, тот должен исчезнуть.
8. Появление (Appear): противоположно исчезновению.
9. Свойство (Property): используется, когда нужно обновить свойство, например за каждого убитого врага увеличивать на единицу количество набранных очков.
10. Сцена (Scene): применяется, когда нужно обновить текущую сцену, например загрузить следующую по завершении предыдущей.
11. Специальное (Custom): позволяет определить специальный код, который будет управлять персонажем.

Примечание

На момент написания этой книги сайт Popfly поддерживал только код на языке JavaScript. Разработчики Popfly собираются обеспечить поддержку .NET-совместимых языков в версии Silverlight 2, но состояние реализации пока неизвестно.

На рис. 2.25 показано поведение, описывающее движение. Читая слева направо, мы видим следующие составные части:

Рис. 2.25. *Новое поведение, описывающее движение*

- Имя поведения «Motion1»
- Событие, возбуждаемое, когда это поведение выполняется
- Фильтр событий, который позволяет указать, когда должен выполняться обработчик события
- Само поведение – в данном случае описывающее движение
- Звуковой эффект, сопровождающий выполнение поведения
- Кнопку Code, позволяющую добавить свой код

Мы создадим поведения для каждого персонажа игры Солдат LEGO и должны будем определить следующие основные аспекты:

- Событие, при возникновении которого следует выполнять поведение
- Собственно поведение, или действие, предпринимаемое при возникновении события

Переименование поведений

Чтобы переименовать поведение, щелкните по его текущему имени – это имя будет помещено в редактируемое текстовое поле, как показано на рис. 2.26.

Рис. 2.26. *Переименование поведения*

Создание поведений главной сцены

Для начала мы создадим набор поведений главной сцены. Нам нужно, чтобы область просмотра перемещалась вместе с главным персонажем, поскольку тот всегда должен оставаться видимым.

Использование ярлыков для создания поведений главной сцены

Находясь в конструкторе сцены, нажмите кнопку Behaviors (рис. 2.16). Вы попадете в окно Main Scene Behaviors, показанное на рис. 2.27. Раскройте список Shortcuts в правом верхнем углу и выберите пункт have a viewport following Main 1 (перемещать область просмотра вместе с Main 1). Этот ярлык определяет поведение, при котором область просмотра (видимая часть сцены) следует за персонажем Main: если персонаж движется вправо, то и область просмотра смещается в том же направлении, открывая новые части сцены.

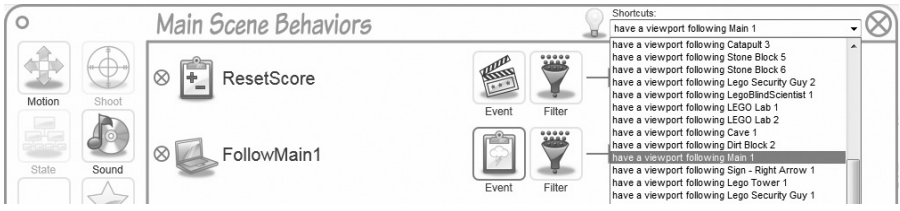

Рис. 2.27. *Добавление на главную сцену поведения с помощью ярлыков*

Создание поведения, описывающего изменение свойства, для сброса состояния здоровья

Нам необходимо создать поведение, которое будет сбрасывать в начальное состояние свойство HealthBar, определенное выше. Щелкните по значку Property на панели инструментов слева, чтобы добавить поведение, описывающее изменение свойства (рис. 2.28).

Рис. 2.28. *Добавление поведения, описывающего изменение свойства*

Мы хотим записать в свойство HealthBar значение 100 в момент загрузки игры. Поскольку Popfly Game Creator по умолчанию возбуждает событие при загрузке сцены, нам не нужно самостоятельно указывать, когда событие должно сбрасывать счет. Чтобы записать в HealthBar значение 100, нажмите кнопку Property, выберите свойство Health, установите переключатель в положение Set it to: (Установить в) и введите в текстовое поле число 100, как показано на рис. 2.29.

Рис. 2.29. *Присваивание свойству Health значения 100 при загрузке сцены*

Создание специального поведения

И напоследок мы добавим специальное поведение, которое заставит персонаж HealthBar скользить вместе с областью просмотра. Для этого нажмите кнопку Custom на панели инструментов – в списке поведений сцены Main появится новое поведение типа Code (рис. 2.30).

Рис. 2.30. *Создание специального поведения главной сцены*

Нажмите кнопку Event и укажите, что нас интересует событие изменения свойства, которое должно возникать при каждом изменении X (рис. 2.31). Это событие будет возникать при перемещении главного персонажа вдоль оси X.

Рис. 2.31. *Задание события изменения свойства, вызывающего исполнение написанного нами кода при перемещении главного персонажа вдоль оси X*

Определив, когда должно возбуждаться событие, мы должны написать код его обработчика. Для этого нажмите кнопку Code, показанную на рис. 2.30, и введите JavaScript-код, приведенный в примере 2.3.

Пример 2.3. Код автоматического перемещения персонажа HealthBar

```
//Оригинальный код взят с: http://www.popfly.com/users/nick-potter/
U-Boot%20Treasure%20Hunt

// Получить текущее положение области просмотра
var viewportX = Game.CurrentScene.GetValue("X");

// если viewport отрицательно, сбросить положение
// индикатора здоровья в 0
if(viewportX < 0)
{
  viewportX = 0;
}

// Получить ссылку на персонаж healthBar
// и установить значение свойства
var healthBar = Game.CurrentScene.GetActor("HealthBar 1");
healthBar.SetValue("X",viewportX);
```

В этом коде мы сначала получаем смещение области просмотра Viewport вдоль оси X, пользуясь встроенными в Popfly функциями на языке JavaScript. Затем мы проверяем, что смещение неотрицательно, получаем ссылку на экземпляр персонажа «HealthBar 1» и устанавливаем его смещение вдоль оси X равным смещению Viewport. Иными словами, персонаж «HealthBar 1» движется синхронно с областью просмотра, все время оставаясь видимым. Если главный персонаж сместит область просмотра в точку x=100, то и «HealthBar 1» окажется в этой точке.

Примечание

На самом деле механизмы графической IDE Popfly просто вызывают JavaScript-функции, определенные в Popfly Game Creator API. Это означает, что и вы, как разработчик, можете пользоваться этими функциями при реализации специальных поведений. Дополнительную информацию о Popfly Game Creator API см. на странице *http://www.popflywiki.com/GameCreatorAPI.ashx*.

Добавление поведений главного персонажа с помощью ярлыков

На предыдущем шаге мы добавили поведения самой *сцены*; теперь добавим поведения *персонажей*, начав с персонажа Main. Перейдите в конструктор персонажей, выберите персонажа Main и нажмите кнопку Behaviors. Для начала добавим поведения для перемещения влево/вправо и прыжков. Проще всего воспользоваться ярлыком run and jump by pressing keys (side view) (бежать и прыгать по нажатиям клавиш (вид сбоку)), который показан на рис. 2.32.

На рис. 2.33 видно, что в результате этой операции автоматически было добавлено сразу несколько поведений главного персонажа.

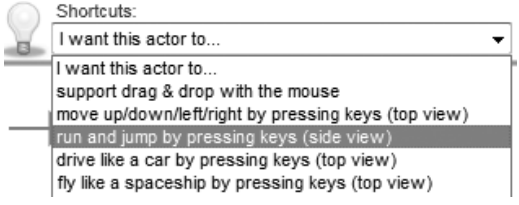

Рис. 2.32. *Добавление поведений главного персонажа, описывающих перемещение и прыжки, с помощью выбора ярлыков*

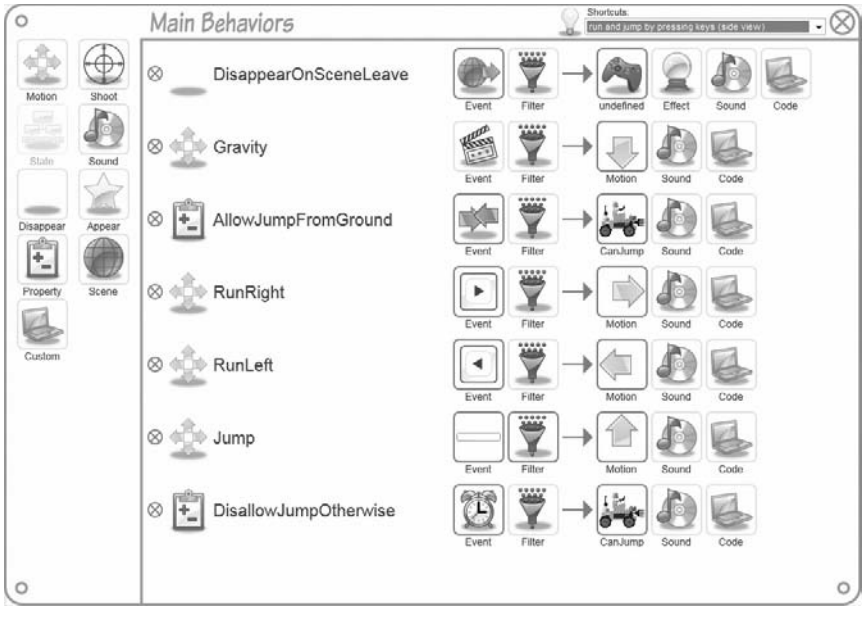

Рис. 2.33. *Список вновь добавленных поведений главного персонажа*

Объясним, что они означают.

DisappearOnSceneLeave

Этим поведением по умолчанию наделяются все персонажи. Когда персонаж покидает сцену, он исчезает.

Gravity

Непрерывное движение вниз. Это поведение действует на персонажа, как сила земного притяжения.

AllowJumpFromGround

Записывает в свойство персонажа `CanJump` значение 1, когда персонаж стоит на твердой поверхности.

RunRight

Перемещает персонажа вправо при нажатии клавиши со стрелкой вправо.

RunLeft

Перемещает персонажа влево при нажатии клавиши со стрелкой влево.

Jump

Заставляет персонажа прыгнуть вверх при нажатии пробельной клавиши. Это событие возбуждается только тогда, когда свойство `CanJump` устанавливается в 1 из поведения `AllowJumpFromGround`.

DisallowJumpOtherwise

Это поведение сбрасывает свойство `CanJump` в 0 в каждом кадре. Его действие отменяется описанным выше поведением `AllowJumpFromGround`. Идея в том, что персонаж может прыгать, только стоя на твердой поверхности.

Нам хотелось бы увеличить скорость главного персонажа, когда он движется вправо. Для этого выберите поведение RunRight и нажмите кнопку Motion, чтобы изменить характер перемещения (рис. 2.34).

Рис. 2.34. *Поведение RunRight главного персонажа*

В результате появляется конструктор движений, позволяющий уточнить поведение персонажа (рис. 2.35). Слева находится «компас», показывающий, что мы будем двигаться на восток относительно сцены. Поскольку мы хотим, чтобы машинка ехала быстрее, увеличьте ускорение (acceleration) с 200 до 500. В разделе Continue Moving (Продолжать движение) устанавливается, что машинка будет ехать, пока нажата клавиша со стрелкой вправо, которая сопоставлена с поведением RunRight. Поэкспериментировав с заданием разных параметров, вы можете посмотреть, как это будет выглядеть, нажав кнопку Preview (Предварительный просмотр), – она запустит игру с новыми параметрами.

Создание поведения катапульты

Последнее поведение главного персонажа – подбрасывание его в воздух после запрыгивания на персонаж Catapult. Как вы, наверное, догадались, поведение катапульты заключается в том, чтобы подбросить главного персонажа, когда тот соприкасается с верхней платформой

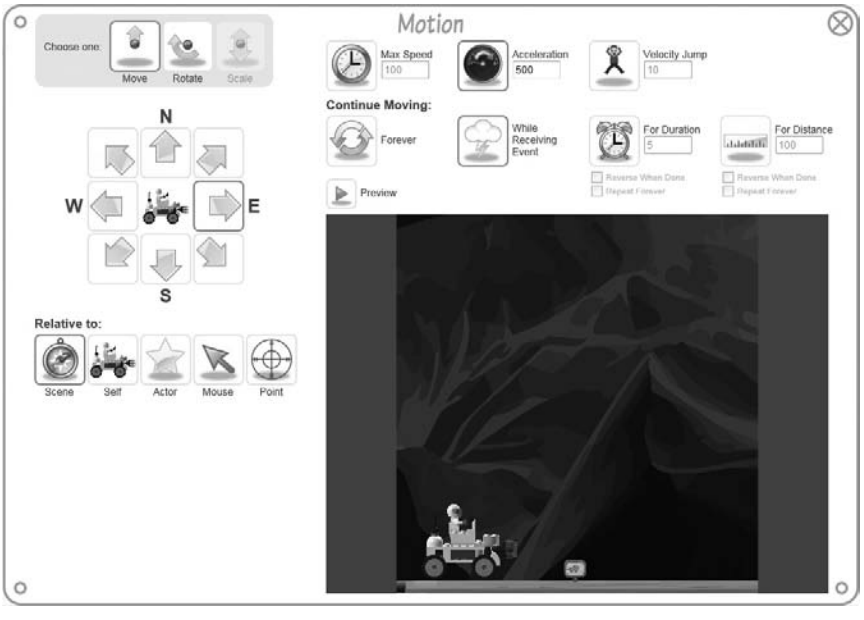

Рис. 2.35. *Увеличение ускорения при движении главного персонажа вправо*

Рис. 2.36. *Новое поведение, описывающее движение*

катапульты. Для этого создайте пустое поведение, нажав кнопку Motion в списке поведений (рис. 2.36).

Затем, выбрав поведение Motion1, нажмите кнопку Event, чтобы задать событие, активирующее это поведение. Мы хотим, чтобы оно активировалось, когда персонаж Main наступает на персонажа Catapult. Следовательно, речь идет о событии столкновения (Collision), параметры которого показаны на рис. 2.37.

На рис. 2.37 вы видите два значка звездочки между словами во фразе Raise this event when actor and actor collide (Возбуждать это событие, когда персонаж и персонаж сталкиваются). Щелкните по любой звездочке (безразлично какой) и установите переключатель в положение Myself (the current instance of Main) (Я (текущий экземпляр Main)), чтобы определить одного из сталкивающихся персонажей (рис. 2.38).

Для определения другого персонажа установите переключатель в положение Any instance of this actor (Любой экземпляр этого персонажа) и щелк-

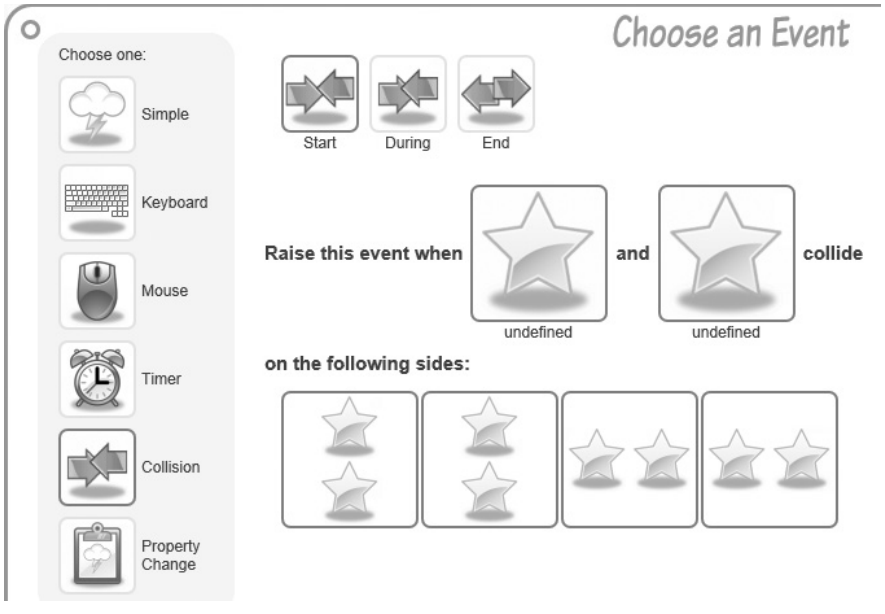

Рис. 2.37. *Создание события столкновения*

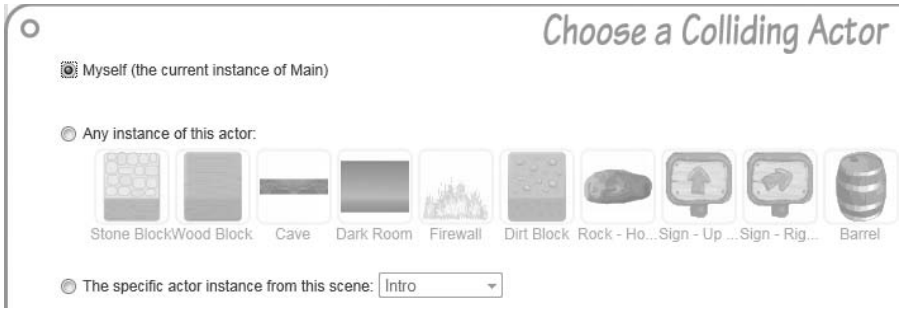

Рис. 2.38. *Задание персонажа в событии столкновения*

ните по персонажу Catapult, как показано на рис. 2.39. В результате будет добавлено событие столкновения между главным персонажем и любой катапультой.

Нам осталось указать, что событие столкновения должно возбуждаться только тогда, когда главный персонаж оказывается сверху катапульты. Для этого нужно мышью отметить то сочетание сторон, при котором возникает событие. Поскольку нас интересует лишь случай, когда главный персонаж стоит на катапульте, то случаи bottom (снизу), right (справа) и left (слева) нужно отключить, щелкнув по ним мышью.

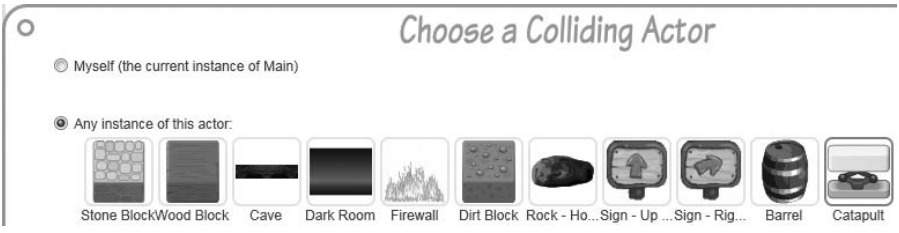

Рис. 2.39. *Выбор любого экземпляра персонажа Catapult*

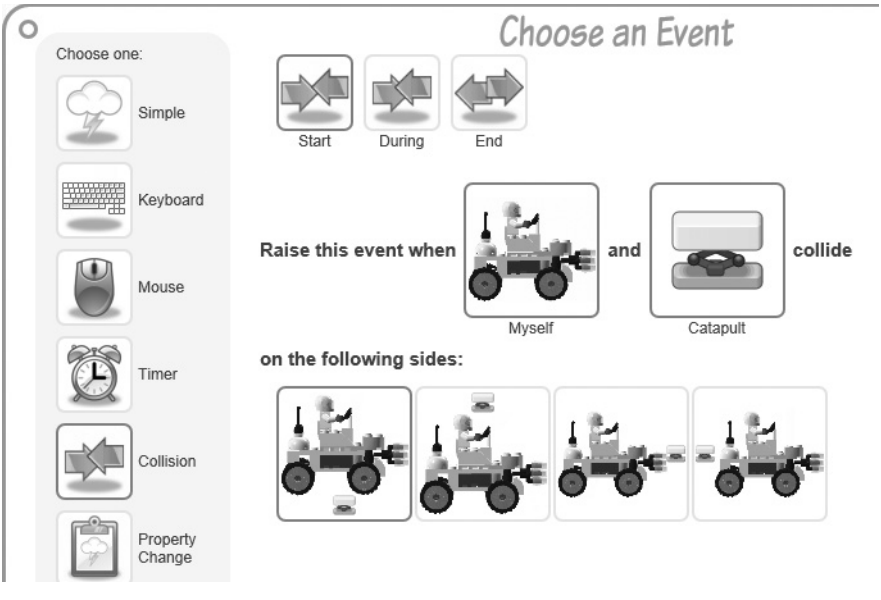

Рис. 2.40. *Окончательная конфигурация события столкновения*

Окончательная конфигурация события столкновения показана на рис. 2.40.

Определив, когда возникает событие столкновения, нужно описать, что в этом случае делать. Для этого выберите поведение «Motion1» и нажмите кнопку Motion (рис. 2.36). Мы хотим определить движение так, чтобы главный персонаж взмывал в воздух, оказавшись на катапульте. Как раз для этой цели в Popfly есть встроенное движение Velocity Jump (Подскок) с эффектом распрямляющейся пружины. Установим свойства этого движения следующим образом:

- задайте направление движения – вверх;
- нажмите кнопку Velocity Jump и задайте скорость 40;

- нажмите кнопку For Duration и установите продолжительность прыжка 0.5 секунды.

Окончательная конфигурация этого поведения показана на рис. 2.41.

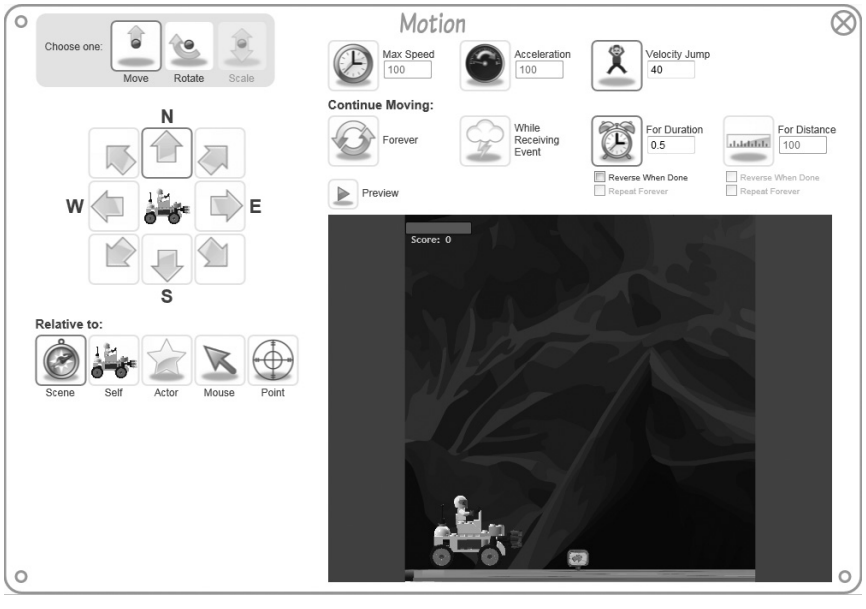

Рис. 2.41. *Окончательная конфигурация поведения катапульты*

Создание поведений охранников LEGO

Поведение персонажа «охранник LEGO» должно быть малопредсказуемым, то есть промежутки времени между выстрелами и прыжками должны быть случайны.

Добавление поведений, описывающих притяжение и прыжки, с помощью ярлыков

Первым делом мы еще раз воспользуемся встроенным ярлыком run and jump by pressing keys (side view), который был описан выше в разделе «Добавление поведений главного персонажа с помощью ярлыков». Он позволит нам автоматически создать поведения, описывающие притяжение и прыжки. Это поможет сэкономить время и избежать ошибок, которые могли бы вкрасться при кодировании вручную. Поскольку персонаж Security будет двигаться случайно, а не по нажатиям клавиш, удалите поведения RunLeft и RunRight. После этого останутся только поведения, показанные на рис. 2.42.

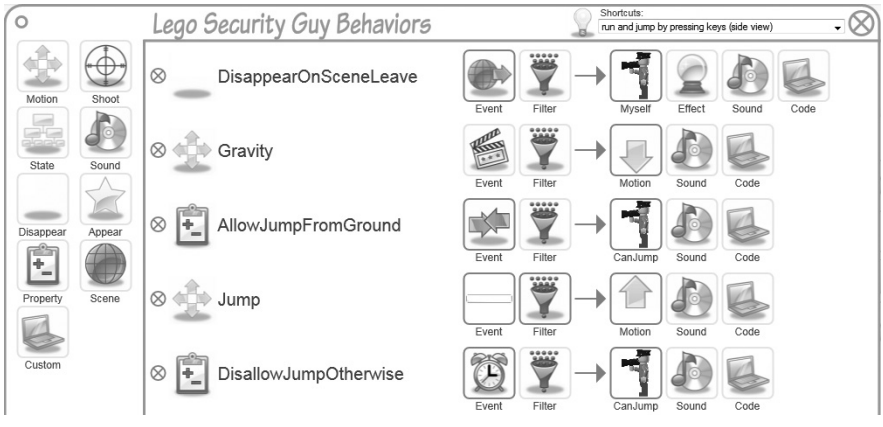

Рис. 2.42. *Использование ярлыков для ускоренного добавления типичных поведений*

Задание случайных прыжков

Нам нужно изменить один аспект готового поведения – событие, активирующее поведение Jump. Сейчас это событие возникает при нажатии пробельной клавиши. Нажмите кнопку Event для события Jump и измените его тип: вместо события клавиатуры укажите случайный промежуток времени длительностью от 2 до 8 секунд, как показано на рис. 2.43. Вы, конечно, поняли, что теперь наш охранник будет подпрыгивать каждые 2–8 секунд.

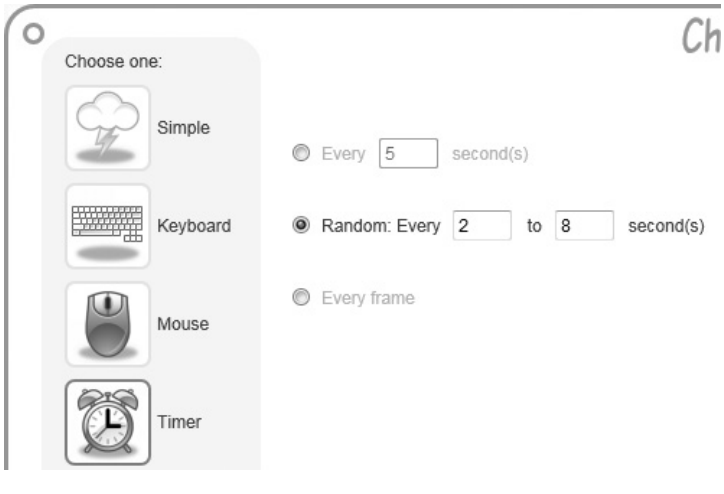

Рис. 2.43. *Конфигурирование события, возникающего через случайные промежутки времени*

Добавление поведения, описывающего случайное блуждание

Теперь создадим еще одно поведение: помимо случайных подпрыгиваний, охранник будет еще ходить вправо-влево через случайные промежутки времени. Для этого создайте новое поведение типа Motion и задайте для его активации такое же событие, как на рис. 2.43.

Затем сконфигурируйте для этого поведения движение: сначала влево (на запад) на протяжении 2 секунд, а затем – в обратном направлении (на восток), как показано на рис. 2.44.

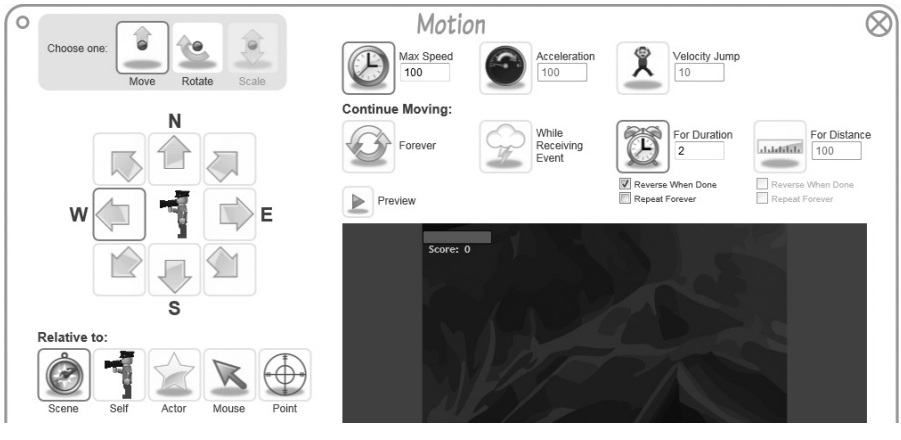

Рис. 2.44. *Конфигурирование движения: влево в течение 2 секунд, затем в обратном направлении*

Добавление поведения, описывающего стрельбу

Следующий шаг – добавить поведение, описывающее стрельбу по главному персонажу. Для этого выберите из списка поведений персонажа Security поведение Shoot, как показано на рис. 2.45.

И на этот раз выберем в качестве активатора стрельбы событие, возникающее через случайные промежутки времени. Для этого щелкните по событию Shoot1, выберите опцию Timer и установите для таймера переключатель Random every 2 to 8 seconds (Случайно каждые 2–8 секунд) (рис. 2.43).

Рис. 2.45. *Новое поведение, описывающее стрельбу*

Затем выберите персонаж Projectile (он же пуля), который понадобится для поведения, описывающего стрельбу. Нажмите кнопку Projectile, чтобы ознакомиться со списком встроенных в Popfly «снарядов» и выберите персонаж «Fire Bullet» (Ружейная пуля) (рис. 2.46).

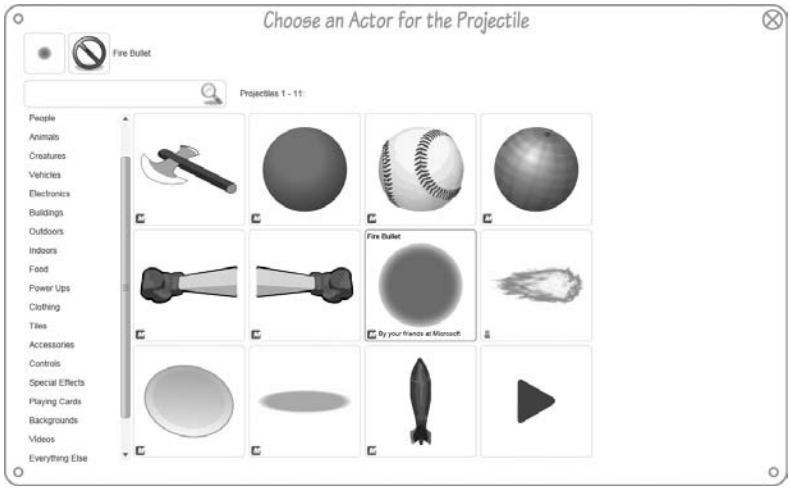

Рис. 2.46. *Выбор ружейной пули в качестве «снаряда»*

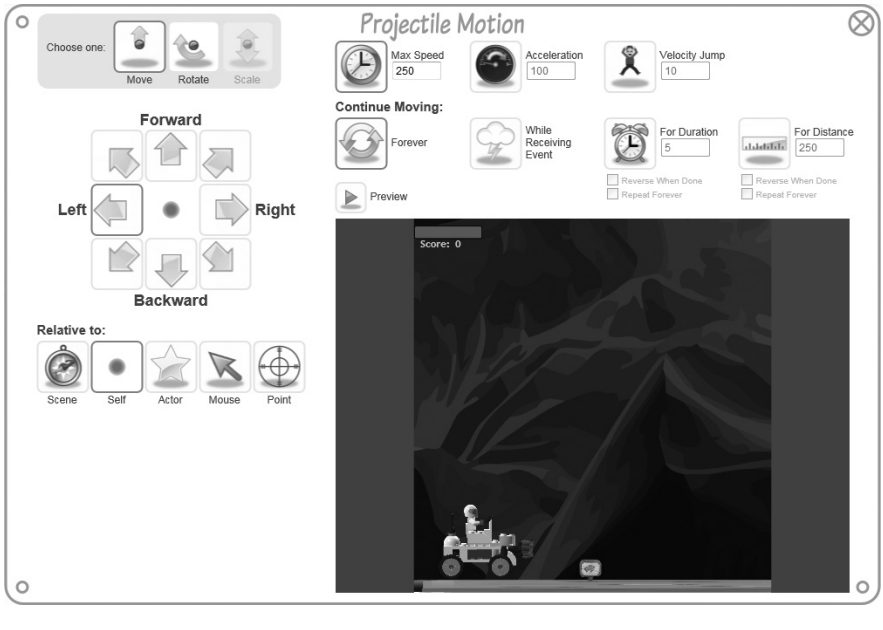

Рис. 2.47. *Конфигурация движения для персонажа Projectile*

Необходимо сконфигурировать поведение, описывающее движение пули. Нажмите кнопку Motion для поведения Shoot1 (рис. 2.44). Мы хотим, чтобы пуля летела слева направо с максимальной скоростью (Max Speed) 250. Соответствующие настройки показаны на рис. 2.47.

Добавление поведения, описывающего исчезновение

Далее нам необходимо поведение, в результате которого охранник «умирает», или исчезает. Добавьте поведение, описывающее исчезновение охранника, как показано на рис. 2.48.

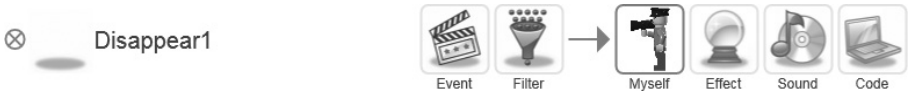

Рис. 2.48. *Поведение, описывающее исчезновение*

Нажмите кнопку Event для поведения Disappear1, чтобы добавить событие столкновения, при возникновении которого охранник должен исчезнуть. Выберите событие типа Collision и назначьте сталкивающихся персонажей: Myself (the current instance of this actor) (Я (текущий экземпляр охранника)) и персонаж Main (рис. 2.49).

Таким образом, охранник исчезает, когда в него врезается главный персонаж.

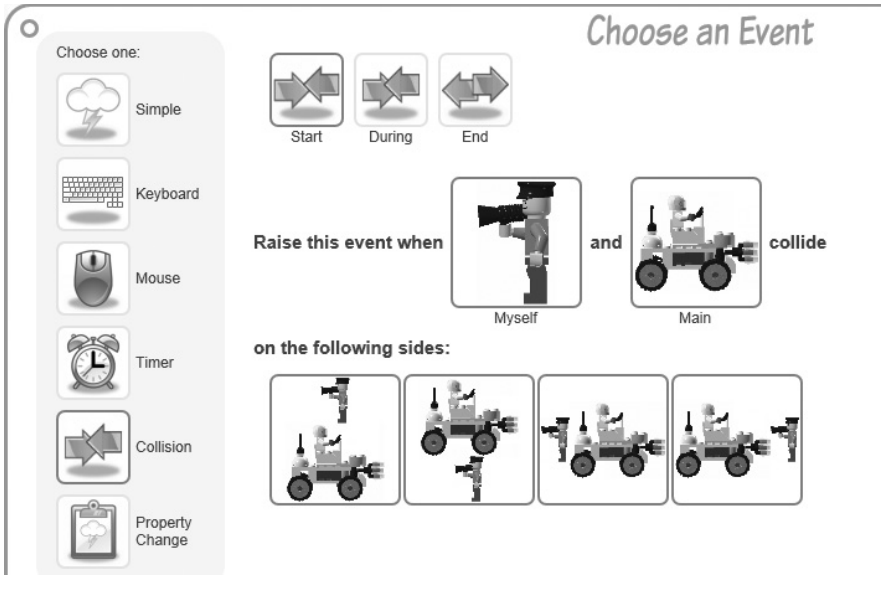

Рис. 2.49. *Событие столкновения, приводящее к исчезновению*

Добавление поведений пули

В нашей игре используется индикатор здоровья, поэтому нужно добавить поведение пули и модифицировать поведения, настроенные в разделе «Добавление поведения, описывающего стрельбу».

Изменение поведения DisappearOnCollision

Сначала изменим существующее поведение DisappearOnCollision. Для этого выберите из списка персонаж Fire Bullet и нажмите кнопку Behaviors, чтобы вывести список его поведений (рис. 2.50).

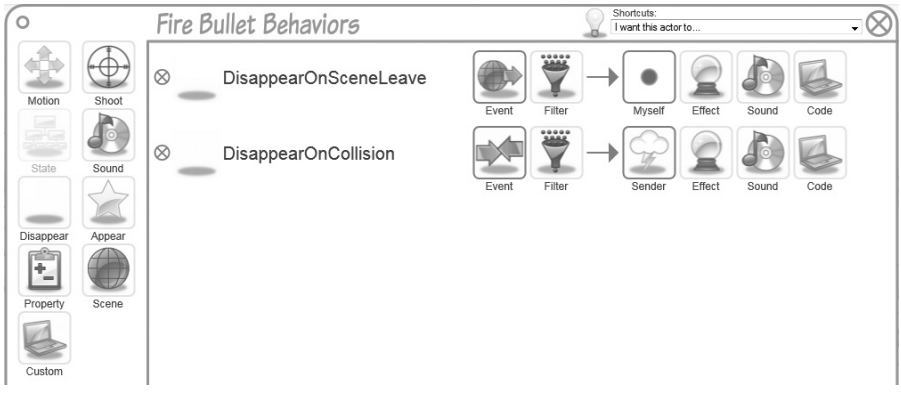

Рис. 2.50. *Стандартные поведения персонажа Fire Bullet*

По умолчанию событие DisappearOnCollision сконфигурировано так, что исчезают оба столкнувшихся персонажа. Это не годится, поскольку со сцены исчез бы и наш главный персонаж, что нежелательно. Мы хотим, чтобы при столкновении исчезла только пуля и одновременно уменьшилось бы здоровье главного персонажа.

Чтобы изменить поведение DisappearOnCollision, нажмите кнопку Sender (Отправитель) (рис. 2.50) и установите переключатель в положение Myself (the current instance of Fire Bullet) (Я (текущий экземпляр Fire Bullet)), как показано на рис. 2.51.

Создание специального поведения, уменьшающего здоровье

Следующая задача – добавить специальное поведение, уменьшающее значение свойства Health. Для этого создайте новое специальное поведение и нажмите кнопку Event, чтобы определить активирующее его событие. В конструкторе выберите событие столкновения (типа Collision) и сконфигурируйте его так, чтобы оно возникало при соприкосно-

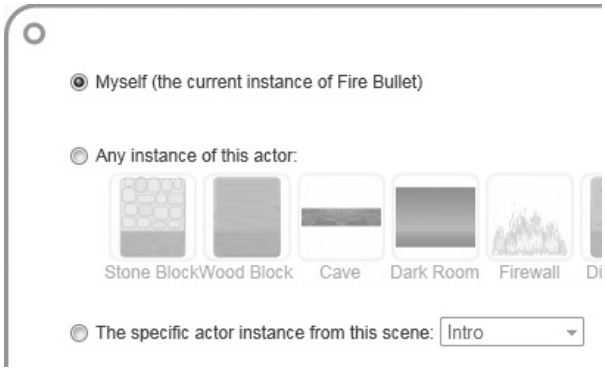

Рис. 2.51. *Изменение поведения, описывающего исчезновение, таким образом, что исчезает текущий экземпляр персонажа Fire Bullet*

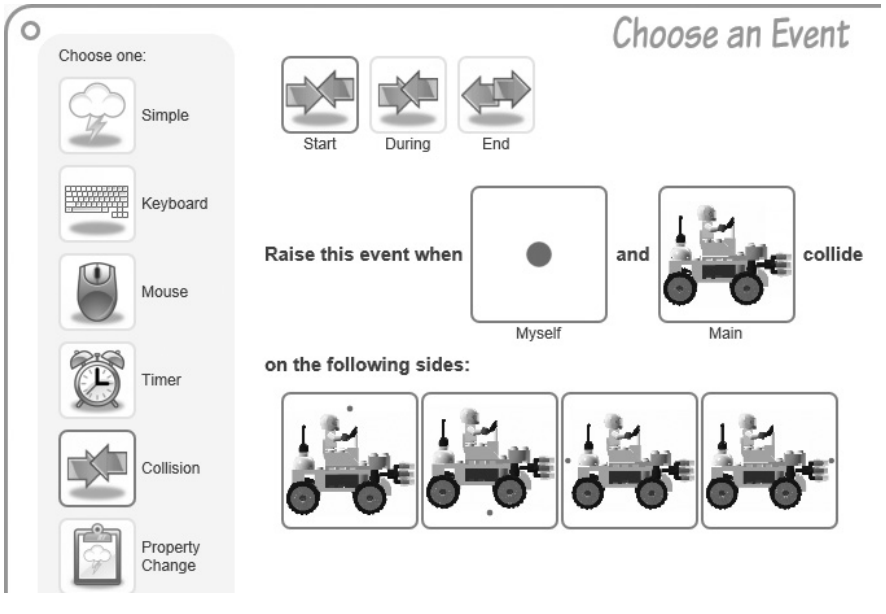

Рис. 2.52. *Конфигурация события, при котором обновляется свойство Health*

вении персонажей Myself (the current instance of Fire Bullet) и Main любыми сторонами (рис. 2.52).

Нажмите кнопку Code и в окно кода введите текст, приведенный в примере 2.4.

Пример 2.4. Персонаж Fire Bullet уменьшает свойство Health на 20

```
Game.SetValue("Health", Game.GetValue("Health") - 20);
```

По завершении окно кода должно выглядеть, как показано на рис. 2.53. Отметим, что здесь же находится окно предварительного просмотра, чтобы можно было протестировать специальное поведение, не загружая игру целиком.

Рис. 2.53. *Код специального поведения персонажа Fire Bullet*

Создание поведения скелета LEGO

Поскольку скелеты LEGO должны вести себя как безмозглые зомби, то мы заставим их бездумно преследовать главного персонажа, как только они оказываются в области просмотра.

Создание поведений, описывающих притяжение и прыжки, с помощью ярлыков

Как и с охранником, воспользуемся встроенным ярлыком run and jump by pressing keys (side view) для автоматического создания поведений, описывающих притяжение и прыжки.

Не забудьте, как и раньше, удалить поведения RunLeft и RunRight.

Изменение поведения, описывающего прыжки

Мы изменим событие, активирующее поведение Jump, так чтобы оно возникало случайно с промежутком от 2 до 8 секунд (см. рис. 2.43).

Создание поведения для преследования главного персонажа

Теперь нам необходимо поведение, которое заставит скелетов бездумно преследовать главного персонажа. Для этого создайте поведение

типа Motion для персонажа Skeleton. Сконфигурируйте активирующее его событие, так чтобы оно возникало, когда персонаж Main входит в область просмотра, где имеется персонаж Skeleton (рис. 2.54).

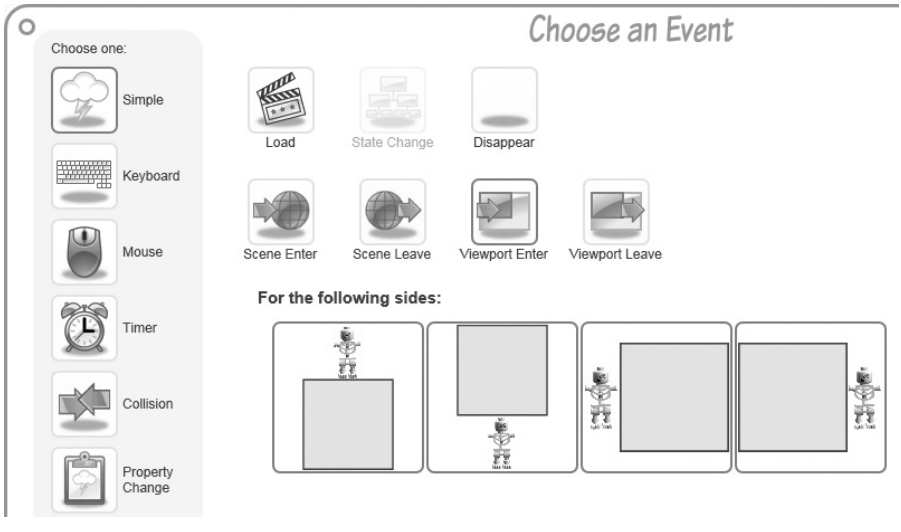

Рис. 2.54. *Возбуждение события в момент, когда главный персонаж входит в область просмотра*

Итак, скелет начинает движение, когда главный персонаж оказывается на одном с ним участке сцены. Теперь нужно определить, что это за движение. Для этого нажмите кнопку Motion для поведения Motion1 и установите для параметра Relative to (Относительно) значение Main (то есть скелет должен двигаться относительно главного персонажа). Обратите внимание, что теперь надписи рядом со стрелками изменились: вместо N и S появились слова Toward (К) и Away (От), как показано на рис. 2.55. Чтобы завершить определение движение, задайте направление Toward (к главному персонажу). При таком определении скелет, оказавшись в одной области просмотра с главным персонажем, начнет двигаться по направлению к нему.

Конфигурация поведения скелета при исчезновении

Далее мы добавим поведение скелета при исчезновении – он будет исчезать, столкнувшись с главным персонажем нижним или правым боком. Создайте новое поведение типа Disappear и в качестве активирующего события задайте столкновение между текущим персонажем Skeleton (Myself) и персонажем Main. Затем отключите значки, соответствующие столкновениям с верхней и левой стороной скелета, как показано на рис. 2.56.

Рис. 2.55. *Конфигурирование поведения таким образом, что скелет движется в направлении главного персонажа*

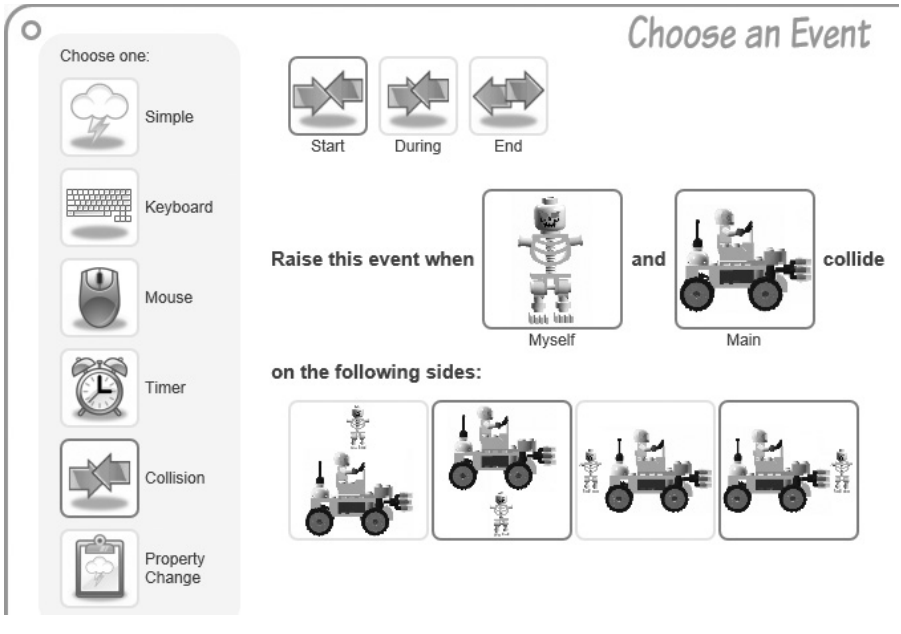

Рис. 2.56. *Конфигурирование события исчезновения таким образом, чтобы оно возникало только, когда персонаж Main оказывается сверху или справа от персонажа Skeleton*

Мы хотим добавить еще и спецэффект в момент столкновения персонажей. Нажмите кнопку Effect для поведения Disappear1 (рис. 2.57).

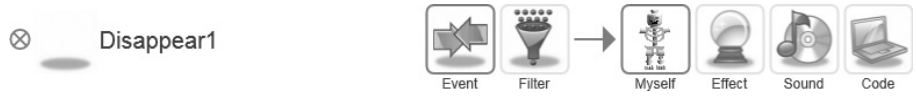

Рис. 2.57. *Поведение Disappear1*

Можно выбрать спецэффект в этом окне или поискать его, введя название в текстовое поле. Наберите в этом поле строку `Flak`, а потом выберите найденный эффект, щелкнув по нему (рис. 2.58). Теперь при исчезновении скелета вы будете видеть клуб дыма.

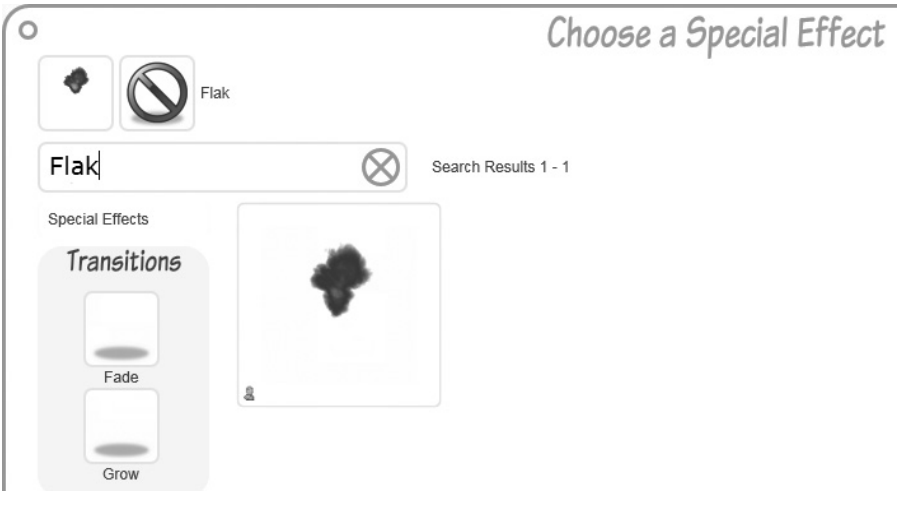

Рис. 2.58. *Выбор спецэффекта Flak*

Создание специального поведения для уменьшения здоровья

Мы хотим наделить скелет специальным поведением, которое, как и в случае с охранником, будет уменьшать здоровье главного персонажа. Только на этот раз речь будет идти не о попадании пули, а о ситуации, когда скелет касается главного персонажа сверху или со спины.

Создайте новое специальное поведение и в качестве активирующего события задайте столкновение между текущим персонажем Skeleton (Myself) и персонажем Main (рис. 2.59). Отключите значки, соответствующие столкновениям с верхней и правой стороной, так чтобы событие возникало только в случае, когда скелет падает на главного персонажа сверху или врезается ему в спину. Не забудьте также включить опцию события `During` (`На всем протяжении`), чтобы главный персонаж

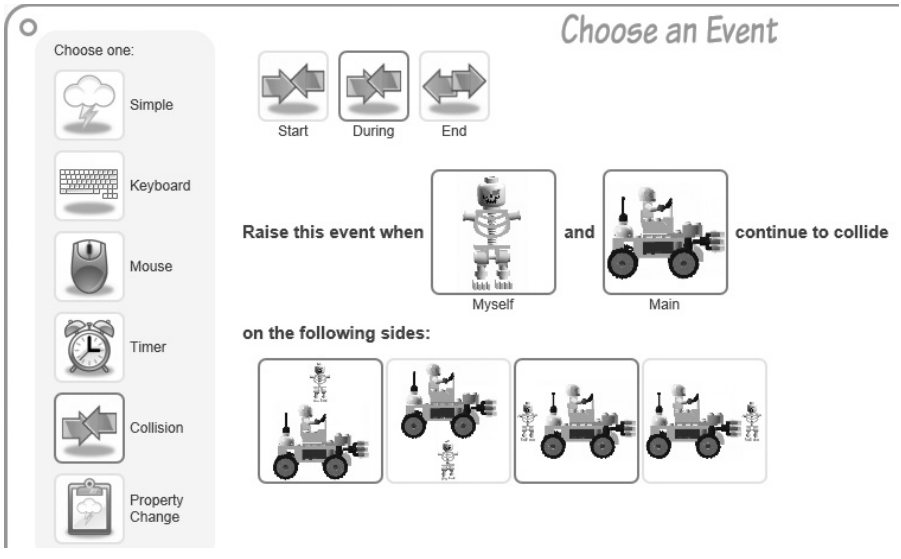

Рис. 2.59. *Это событие столкновения сконфигурировано так, что здоровье главного персонажа продолжает уменьшаться*

продолжал терять здоровье все время, пока находится в контакте со скелетом, находящимся от него сверху или слева.

Поскольку событие столкновения сконфигурировано так, что возникает при любом контакте (рис. 2.59), то во всех кадрах, где скелет и главный персонаж соприкасаются, здоровье неуклонно падает. Но так как столкновения происходят очень часто (в каждом кадре), то мы запрограммируем поведение таким образом, чтобы в каждом кадре здоровье уменьшалось всего на 1 (пример 2.5).

Пример 2.5. Уменьшение здоровья на 1 в каждом кадре

```
Game.SetValue("Health", Game.GetValue("Health") - 1);
```

Так как темп игры, скорее всего, равен около 30 кадров в секунду, это означает что здоровье главного персонажа будет уменьшаться на 30 пунктов каждую секунду.

Добавление поведений бочки

Персонаж Barrel должен вести себя, как пороховая бочка, и взрываться при столкновении со скелетом. Для этого мы сначала добавим ярлык, содержащий поведение притяжения, как делали это для скелета и охранника. Затем нужно будет добавить поведение исчезновения, активируемое событием столкновения, чтобы все скелеты, на которых свалилась бочка, пропадали.

Добавление ярлыка с поведением притяжения для бочки

Как мы уже не раз делали, выберите ярлык run and jump by pressing keys (side view) и удалите все входящие в него поведения, кроме Gravity и DisappearOnSceneLeave (рис. 2.60). В результате бочка будет автоматически падать с платформы, на которой находится (см. выше рис. 2.22).

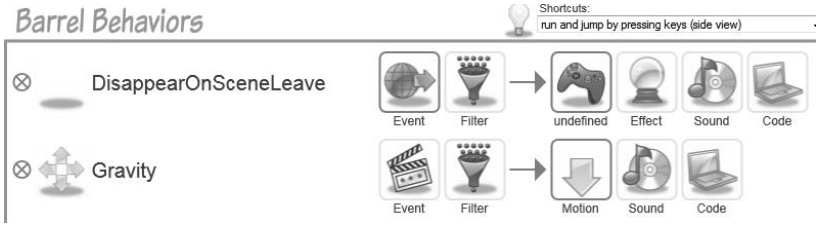

Рис. 2.60. *Удаление всех поведений персонажа Barrel, кроме Gravity и исчезновения при покидании сцены*

Добавление поведения исчезновения

Как и прежде, добавьте новое поведение исчезновения и сконфигурируйте для него событие столкновения между Myself (текущий экземпляр бочки) и скелетом. Оставьте включенными все принимаемые по умолчанию стороны сталкивающихся персонажей, как показано на рис. 2.61.

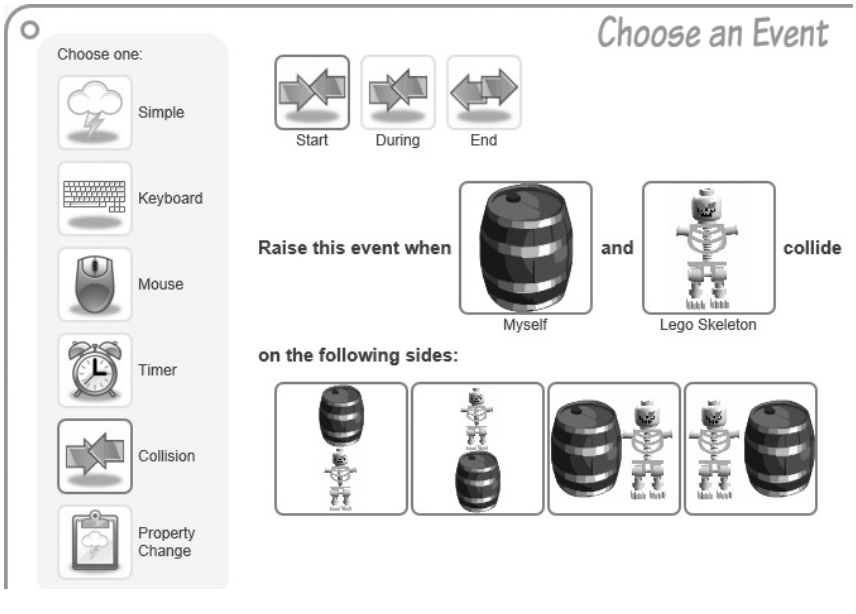

Рис. 2.61. *Конфигурирование события, активирующего исчезновение бочки*

Далее нужно определить, какие персонажи должны исчезать при возникновении этого события. Для этого нажмите кнопку Myself для поведения Disappear1, как показано на рис. 2.62.

Здесь мы отметим переключатель The event sender, поскольку хотим, чтобы исчезли и бочка, и скелет, на который она свалилась (рис. 2.63).

И последний штрих – добавить взрыв при столкновении. Для этого нажмите кнопку Effect для события Disappear1 и выберите эффект «Explosion 2», как показано на рис. 2.64.

Рис. 2.62. *Событие Disappear1*

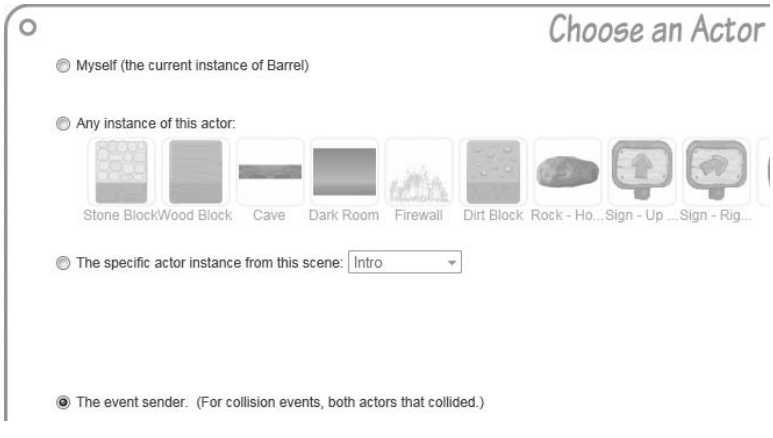

Рис. 2.63. *Вариант «event sender» приводит к исчезновению обоих столкнувшихся персонажей*

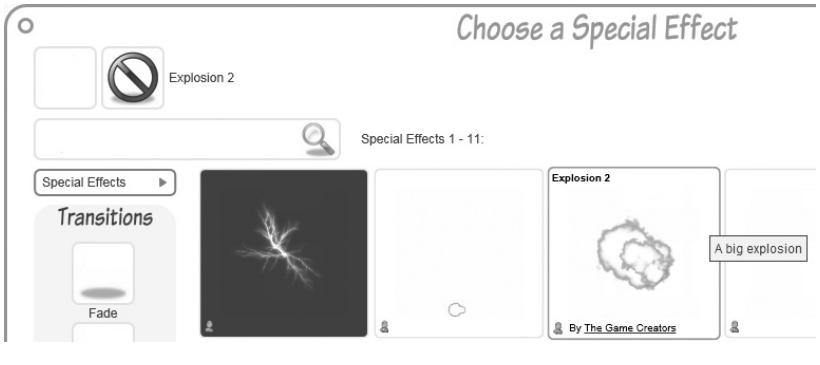

Рис. 2.64. *Эффект взрыва бочки*

Добавление поведения индикатора здоровья

Как вы помните, мы уменьшаем свойство Health в поведениях охранника и скелета. Теперь нужно визуально показать, как изменение этого свойства сказывается на индикаторе здоровья.

Создание специального поведения для изменения свойства HealthValue индикатора здоровья

Щелкните по персонажу HealthBar, перейдите в список его поведений и нажмите кнопку Code для добавления нового специального поведения. В качестве активирующего события выберите изменение свойства (Property Change) и сконфигурируйте его так, чтобы событие возбуждалось при изменении свойства Health (рис. 2.65).

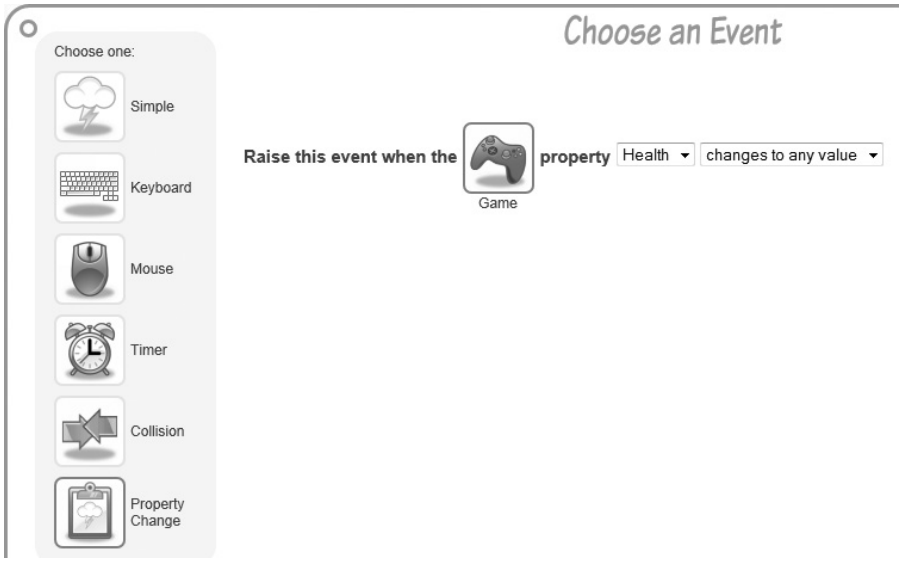

Рис. 2.65. *Событие возбуждается при изменении свойства Health*

Теперь добавьте код специального поведения, приведенный в примере 2.6. Здесь мы получаем ссылку на корень документа Silverlight, находим XAML-объект с именем `HealthValue` и устанавливаем свойство `Width` прямоугольника `HealthValue` равным значению свойства `Health` игры.

Пример 2.6. Изменение ширины прямоугольника HealthValue

```
this.GetVisualRoot().FindName("HealthValue").Width =
  Game.GetValue("Health").toString();
```

Вот какая последовательность событий происходит в игре:

- В начале игры свойство `Health` равно 100 (рис. 2.29).
- Когда главный персонаж (Main) сталкивается с пулей (Fire Bullet), значение `Health` уменьшается на 20.
- В этот момент возбуждается событие изменения свойства `Health` (рис. 2.65), и код из примера 2.6 устанавливает ширину прямоугольника `HealthValue` равной 80 (100 – 20).

На рис. 2.66 показано, как уменьшается ширина прямоугольника `HealthValue` по мере уменьшения свойства `Health`.

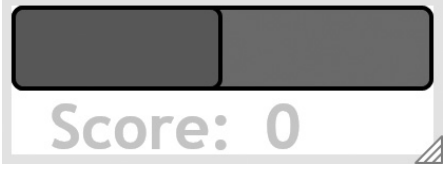

Рис. 2.66. *Визуализация изменения прямоугольника HealthValue при возникновении события HealthPropertyChanged*

Создание поведения смены сцены, завершающего игру

Мы хотим наделить индикатор здоровья еще одним поведением, которое вызывает переход к сцене Lost (Проигрыш), если здоровье достигает нуля. Для этого нажмите кнопку Scene, чтобы добавить поведение смены сцены (рис. 2.67).

Рис. 2.67. *Создание поведения смены сцены*

Затем нажмите кнопку Event, чтобы создать событие, активирующее поведение смены сцены. Мы сконфигурируем событие так, чтобы оно возникало, когда значение `Health` оказывается меньше или равно 0 (рис. 2.68).

Теперь нажмите кнопку Scene (рис. 2.67) – появится окно, где можно выбрать сцену, которая должна стать текущей. Поскольку `Health` равно 0, то есть мы проиграли, то следует перейти к сцене Lost, как показано на рис. 2.69.

Рис. 2.68. *Событие возникает, когда значение Health оказывается меньше или равно 0*

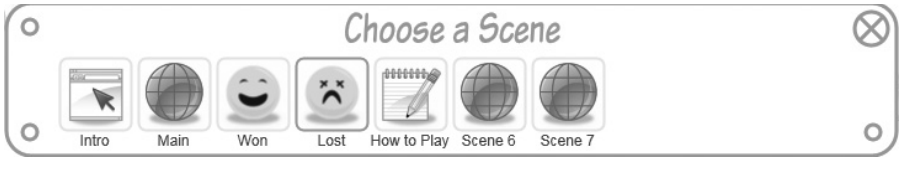

Рис. 2.69. *Поведение смены сцены выбирает сцену Lost*

Добавление поведений Слепого завлаба

Прежде чем добавлять поведения Слепого завлаба, нужно кратко объяснить особенность XAML-разметки, в которой определяется персонаж Blind Scientist. Внутри нее имеется раскадровка Silverlight (анимация по времени), которая приводит к увеличению размеров Слепого завлаба на 400%. Мы собираемся создать поведение, которое запустит раскадровку анимации при столкновении главного персонажа со Слепым завлабом. Но так как мы не хотим, чтобы завлаб увеличивался в четыре раза при каждом столкновении, то заведем булевское свойство `IsHit`, равное true, если анимация уже запускалась ранее.

Добавление специального поведения для запуска раскадровки анимации вырастания

Первым делом создайте специальное поведение для персонажа Blind Scientist и назовите его RunGrowStoryBoard. Сконфигурируйте событие так, чтобы это поведение активировалось при столкновении Слепого завлаба с главным персонажем (рис. 2.70).

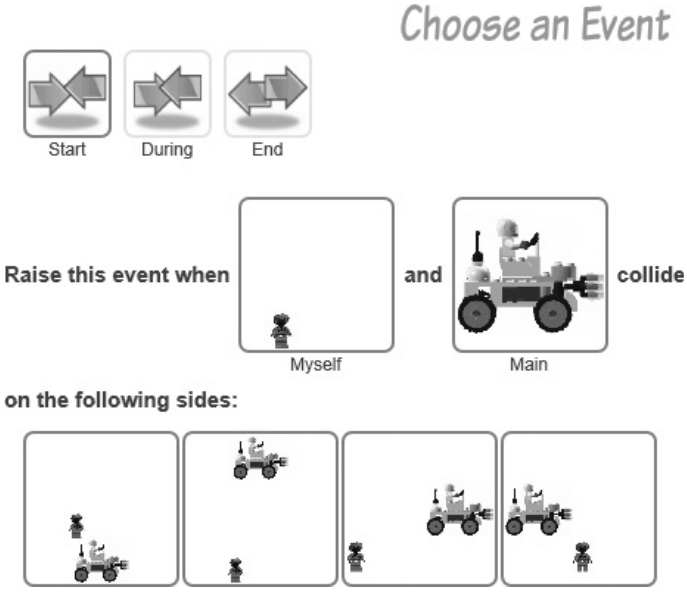

Рис. 2.70. *Событие столкновения, активирующее специальное поведение*

Затем нажмите кнопку Filter для поведения RunGrowStoryBoard, как показано на рис. 2.71.

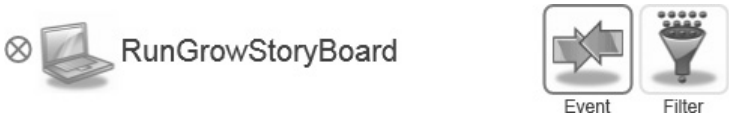

Рис. 2.71. *Добавление фильтра для поведения, активируемого событием*

Мы уже говорили, что фильтр позволяет сделать так, чтобы поведение выполнялось только при соблюдении некоторого условия. В случае поведения RunGrowStoryBoard мы хотим, чтобы раскадровка анимации запускалась лишь при первом столкновении, поэтому в качестве фильтра зададим условие `isHit = 0` (рис. 2.72).

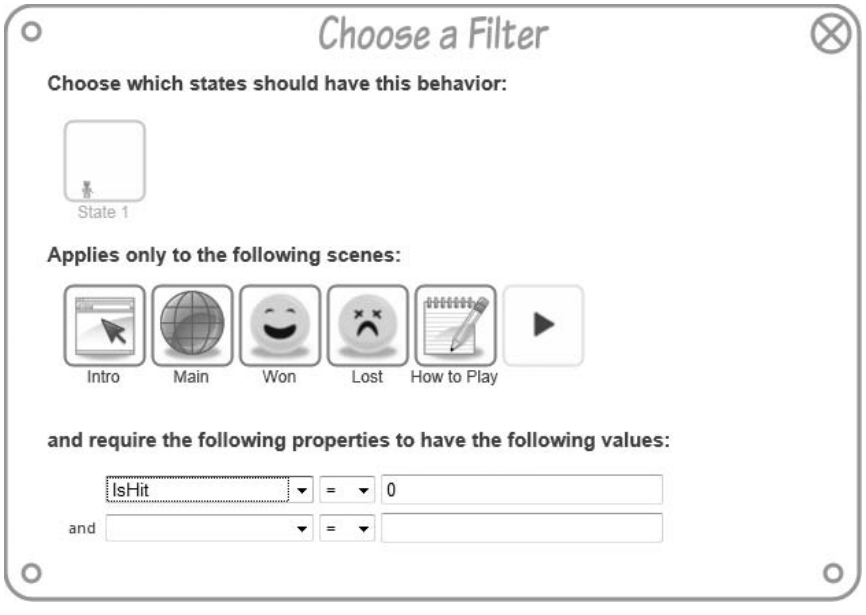

Рис. 2.72. *Задание фильтра, позволяющего выполнить код только в случае, когда свойство isHit равно 0*

Теперь напишем JavaScript-код, который запустит раскадровку и запишет в свойство `IsHit` значение 1 после завершения анимации. Но сначала взгляните на фрагмент XAML-разметки персонажа Blind Scientist (пример 2.7).

Пример 2.7. Фрагмент XAML-разметки персонажа Blind Scientist

```
<Canvas ... Width="500" Height="784" x:Name="canvas">
  <Canvas.Resources>
    <Storyboard x:Name="Grow">
      <!-- Начало анимации -->
...
    </Storyboard>
  </Canvas.Resources>
  <Image Width="125" Height="195" Source="$base$/blindscientist.png"
    Stretch="Fill" Canvas.Top="589" x:Name="image" … >
...
</Canvas>
```

Как видите, в XAML-разметку включено определение раскадровки (Storyboard) с именем `Grow`, которую мы и должны запустить программно. Для этого добавьте код, приведенный в примере 2.8.

Пример 2.8. Вызов раскадровки Silverlight и задание свойства персонажа из JavaScript-сценария

```
this.GetVisualRoot().FindName("Grow").begin();
this.SetValue("IsHit", 1);
```

Здесь мы находим раскадровку `Grow` (см. пример 2.7) и вызываем метод `begin()` объекта `Storyboard`. Он запускает анимацию. Затем мы устанавливаем свойство `IsHit` персонажа Blind Scientist в 1, чтобы отметить тот факт, что анимация вырастания уже состоялась.

Добавление поведений случайного перемещения влево и вправо

Учитывая, что завлаб слеп, его перемещения должны быть хаотичными. Поэтому мы определим два поведения движения: одно влево, другое вправо. Как и раньше, в качестве активирующего события выберем истечение случайного промежутка времени длительностью от 2 до 8 секунд (см. рис. 2.43 выше).

Далее, требуется, чтобы случайные движения начались только после четырехкратного увеличения размеров. Для этого мы воспользуемся фильтром событий. Мы знаем, что после анимации вырастания в свойство `IsHit` записывается 1, поэтому поведения движения влево/вправо можно запускать, когда `IsHit` равно 1 (рис. 2.73).

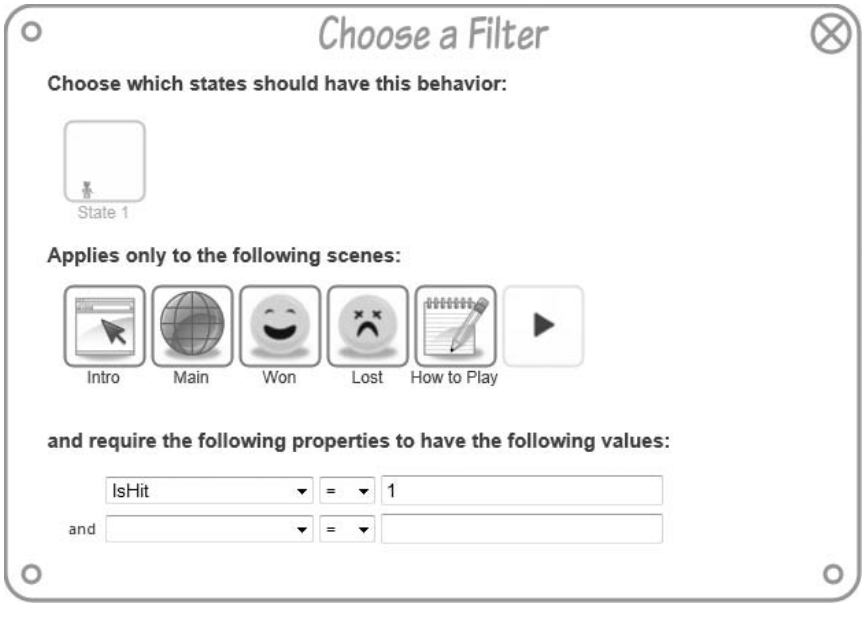

Рис. 2.73. *Добавление фильтра, управляющего моментом запуска поведений, описывающих движение влево/вправо*

Для поведения, описывающего движение влево, установим направление W, максимальную скорость 400 и продолжительность 1.5 секунд (рис. 2.74). Для движения вправо зададим такие же параметры, только направление будет равно E.

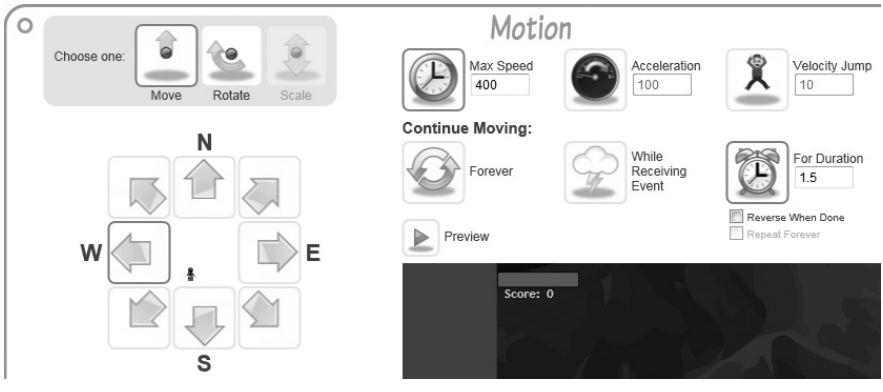

Рис. 2.74. *Конфигурирование поведения, описывающего движение Слепого завлаба*

Добавление поведения, уменьшающего здоровье

Как и для скелета, определим специальное поведение, которое будет уменьшать здоровье главного персонажа при столкновении со Слепым завлабом. Добавьте новое специальное поведение, активируйте его при возникновении события столкновения и задайте для этого события опцию During, чтобы здоровье продолжало уменьшаться, пока два персонажа соприкасаются. Далее, поскольку, прыгая на Слепого завлаба сверху, мы его уничтожаем, выключите кнопку, соответствующую столкновению, при котором главный персонаж оказывается над завлабом. Окончательная конфигурация события столкновения показана на рис. 2.75.

Как и прежде, добавим код, уменьшающий здоровье на 1 в каждом кадре, в котором Слепой завлаб и главный персонаж соприкасаются (пример 2.9).

Пример 2.9.Уменьшение здоровья на 1

```
Game.SetValue("Health", Game.GetValue("Health") - 1);
```

Добавление поведения смены сцены при выигрыше

Чтобы выиграть, главный персонаж должен прыгнуть на голову Слепого завлаба сверху. В этот момент мы хотим показать сцену Won (Выигрыш). Для этого создайте еще одно поведение смены сцены для Слепого завлаба. Оно будет активироваться по событию столкновения между

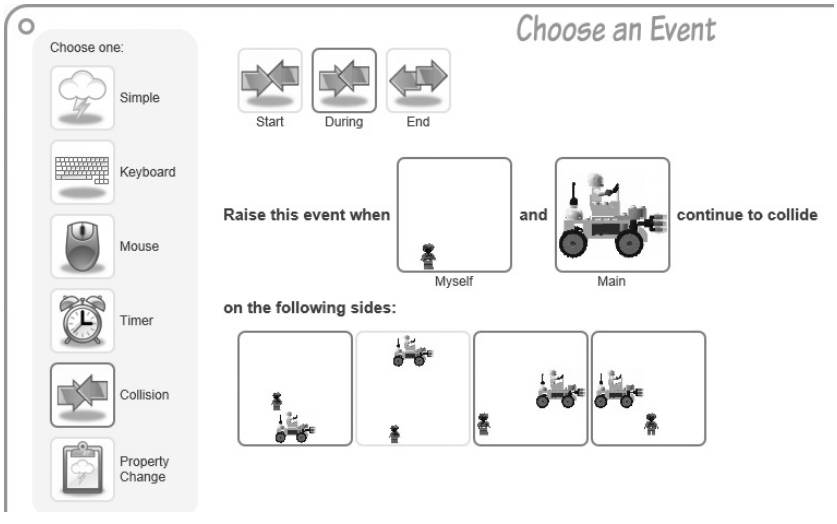

Рис. 2.75. *Уменьшение свойства HealthValue при столкновении двух персонажей*

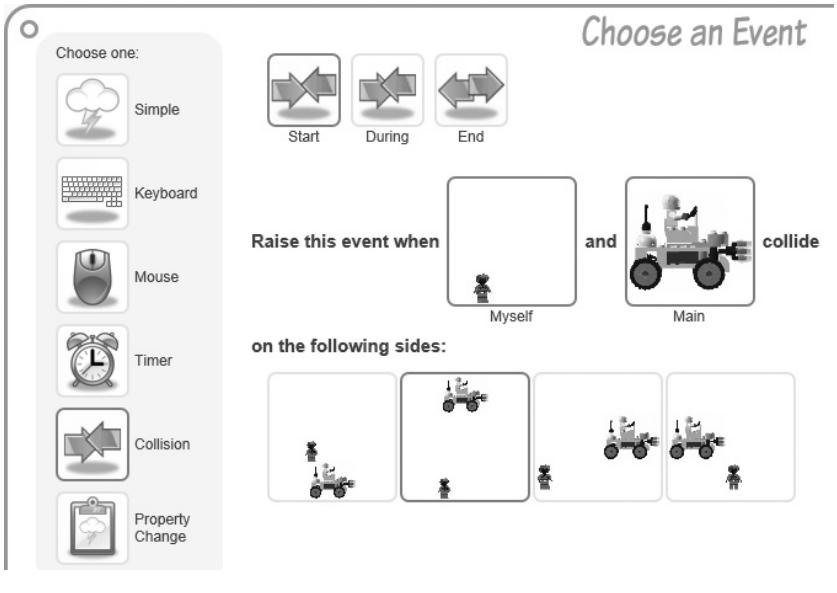

Рис. 2.76. *Конфигурирование события столкновения, приводящего к выигрышу*

завлабом и главным персонажем, как показано на рис. 2.76. Не забудьте, что это событие должно возникать, только когда главный персонаж опускается на Слепого завлаба сверху.

Нужно еще добавить для этого события фильтр, позволяющий выиграть только после того, как свойство `IsHit` стало равным 1. Для этого нажмите кнопку Filter и задайте фильтр так, чтобы событие возникало лишь в случае, когда `IsHit` равно 1 (см. рис. 2.73).

Теперь нажмите кнопку Scene и выберите сцену Won (рис. 2.77), знаменующую успешное окончание игры! Сцены Won и Lost важны, так как Popfly автоматически ведет учет выигрышей и проигрышей пользователей и выводит список лучших игроков.

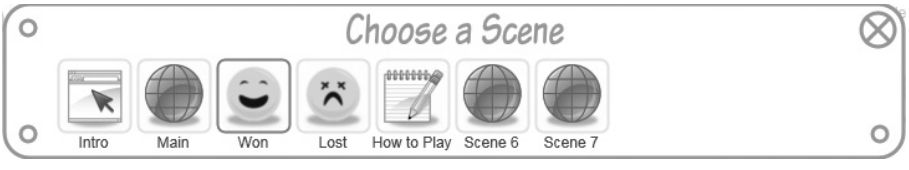

Рис. 2.77. *Переход к сцене Won после того, как главный персонаж прыгнул на Слепого завлаба сверху*

Публикация игры

Когда вы доведете игру до нужного состояния и захотите поделиться ею с окружающими, нажмите кнопку Share в правом верхнем углу. Тем самым вы предоставите доступ к своему проекту всем пользователям Popfly, дадите возможность встроить игру в блог, поместить на сайте Facebook и даже на боковой панели Windows Vista. После опубликования проекта появится всплывающее окно (рис. 2.78), в котором нахо-

Рис. 2.78. *Публикация игры*

дится ссылка на вашу игру, код для встраивания, ссылка на адрес мини-приложения на боковой панели (Sidebar Gadget), ссылка на сборку игры в виде приложения для Facebook и ссылки для рекламирования игры на сайтах Digg, Reddit или по электронной почте.

Публикация на сайте Facebook

Чтобы опубликовать игру на сайте Facebook, щелкните по значку Facebook («f»), который изображен на рис. 2.78. Затем задайте параметры публикации игры, как показано на рис. 2.79.

Рис. 2.79. *Параметры публикации на сайте Facebook*

Щелчок по нижней ссылке Create Your Own Facebook Application (Создать свое Facebook-приложение) отправит вас на страницу с пошаговыми инструкциями по публикации игры (рис. 2.80).

Присвойте приложению имя «LegoSoldier» и сконструируйте уникальный URL своей игры на сайте Facebook. На рис. 2.81 этот URL имеет вид *http://apps.facebook.com/LegoSoldier/*.

Примечание

Если вы хотите протестировать приложение перед тем, как дать к нему общий доступ, отметьте в форме на рис. 2.81 флажок, позволяющий получить доступ к приложению только разработчикам. Тогда приложение будет опубликовано, но закрыто для всех, кроме разработчиков.

Задав на шаге 3 (рис. 2.80) все свойства в форме нового приложения (рис. 2.81), нажмите кнопку Submit, чтобы отправить его серверу. Теперь

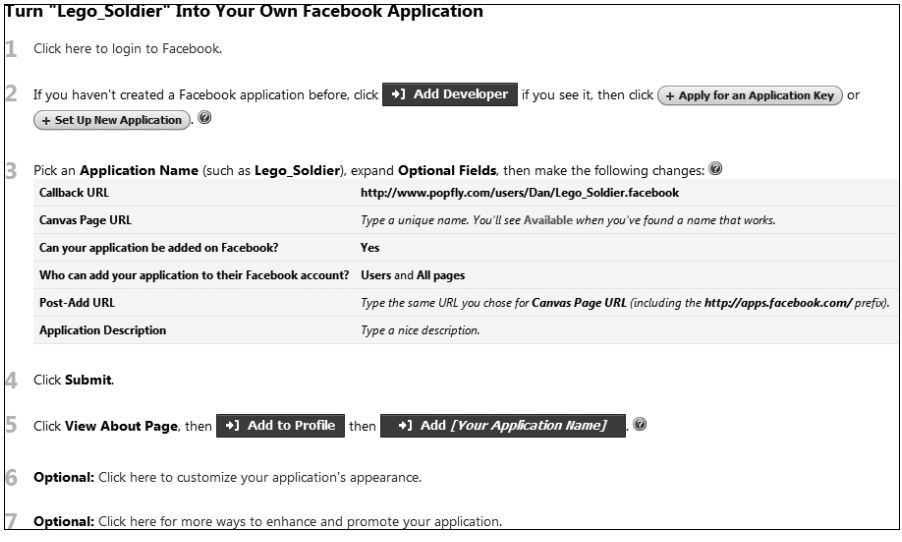

Рис. 2.80. *Popfly включает пошаговые инструкции по публикации игры на сайте Facebook*

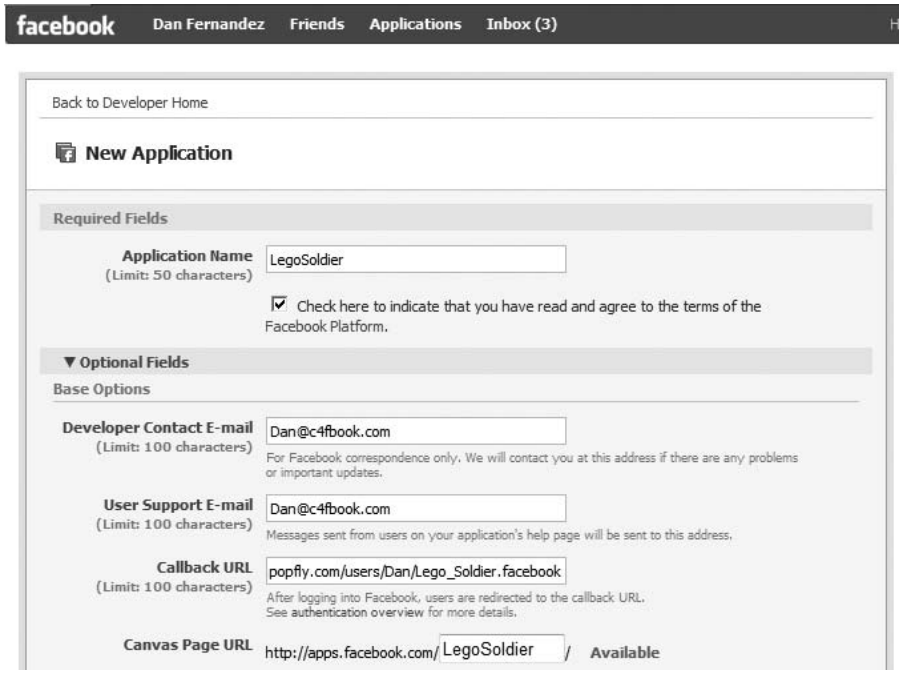

Рис. 2.81. *Конфигурирование игры в виде Facebook-приложения*

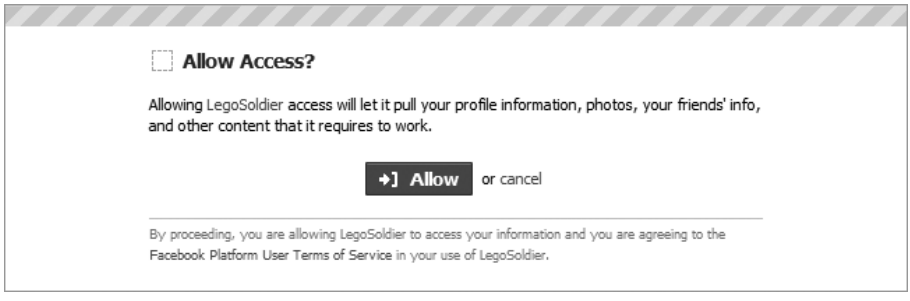

Рис. 2.82. *Запуск Facebook-приложения LegoSoldier*

вы можете напрямую зайти на страницу своего приложения на сайте Facebook по URL страницы Canvas Page, который был задан на рис. 2.81. Если вы устанавливаете это приложение в первый раз, будет задан вопрос, хотите ли вы разрешить к нему доступ (рис. 2.82).

Разрешив доступ к приложению, вы сможете играть непосредственно на сайте Facebook, как показано на рис. 2.83!

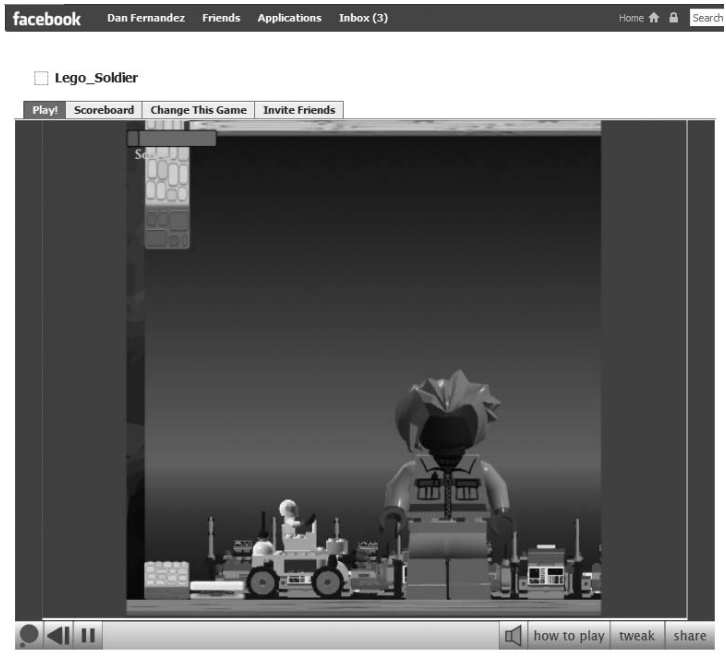

Рис. 2.83. *Играем в LegoSoldier на сайте Facebook*

Заключительные замечания

Солдат LEGO – относительно короткая игра, но на ее примере продемонстрирован ряд важных идей, позволяющих строить стабильно работающие игры: создание персонажей, конструирование сцен, наделение врагов зачатками искусственного интеллекта с помощью случайных перемещений, добавление специальных поведений индикатора здоровья, запуск раскадровки Silverlight и даже публикация на сайте Facebook. Очень здорово, что Popfly позволяет без труда копировать персонажей, сцены, звуковые эффекты и даже целые игры, созданные кем-то другим. Поэтому вы можете взять Солдата LEGO в качестве отправной точки и модифицировать его, как душа пожелает.

3

Чтение новостей: встраиваемый модуль чтения RSS-лент для World of Warcraft

Автор	Габор Ратки и Дэн Фернандес
Сложность	Средняя
Необходимое время	2 часа
Стоимость	$19.99 за *World of Warcraft*; ежемесячная плата $14.99
Программное обеспечение	*World of Warcraft*, AddOn Studio для *World of Warcraft* версия 2.0, Visual C# 2008 Express и Visual Basic 2008 Express или более полная, Internet Explorer версии 7 или выше
Оборудование	Не требуется
Адрес в Интернете	*http://www.c4fbook.com/wowfeedreader*

World of Warcraft производства компании Blizzard Entertainment – без сомнения, самая популярная массовая многопользовательская онлайновая ролевая игра (massively multiplayer online role-playing game – MMORPG). По некоторым оценкам, у нее примерно 10 миллионов подписчиков, и это достижение даже занесено в Книгу рекордов Гиннеса. Конечно, впечатляет, что игра, разворачивающаяся в фантастическом мире Азерот, может приковать вас к компьютеру на долгие часы, но нас больше интересует тот факт, что пользовательский интерфейс поддается модификации и допускает подключение так называемых надстроек (addon), которые упрощают или улучшают взаимодействие игрока с окружающим миром и другими персонажами. А иногда надстройки несут с собой совершенно новую функциональность.

Feed Reader – это надстройка над *World of Warcraft*, которая добавляет возможность читать последние новости, сообщения в форумах, блоги, результаты спортивных состязаний и т. д. – и все это, не покидая комфортный мир Азерота!

Краткий обзор

Прежде чем с головой погрузиться в механизм работы надстройки Feed Reader, дадим краткий обзор ее возможностей.

Запуск программы Feed Grabber

Поскольку надстройка не может обращаться к данным вне самой игры, мы разработаем также .NET-приложение Feed Grabber (рис. 3.1), которое находится в панели задач и преобразует данные из RSS-лент в сценарий на Lua – язык, на котором пишутся надстройки над *Warcraft*.

Рис. 3.1. *Feed Grabber в панели задач*

Feed Reader

После того как Feed Grabber перевел RSS-ленты на язык Lua, мы можем запустить *World of Warcraft* и убедиться, что надстройка Feed Reader (рис. 3.2) действительно отображает RSS-ленты, так что для чтения заголовков BBC даже не придется выходить из игры!

Команды Feed Reader

Feed Reader поддерживает также текстовые команды, позволяющие показать или скрыть окно надстройки, не выходя из окна чата *Warcraft* (рис. 3.3).

Перед тем как приступить к работе

В этой главе предполагается, что вы умеете разрабатывать программы для .NET (на Visual Basic или C#), играете в *World of Warcraft* и, возможно, даже пользовались какими-нибудь надстройками над *Warcraft*, но никогда не писали их сами. Между разработкой надстроек над *Warcraft* и .NET-приложений есть много общего (см. табл. 3.1).

Рис. 3.2. *Feed Reader работает внутри игры*

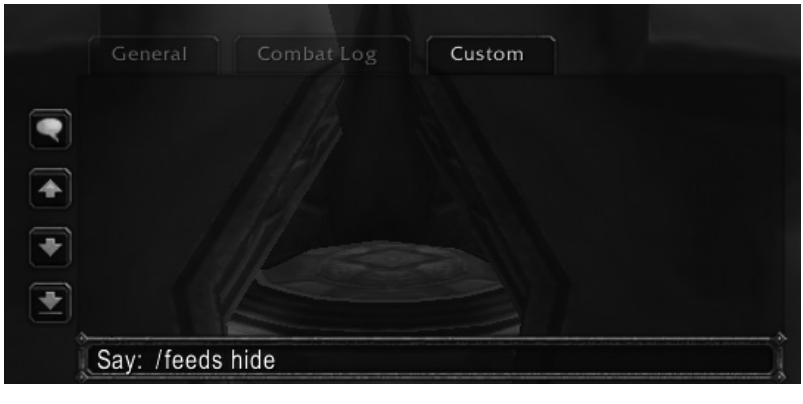

Рис. 3.3. *Текстовая команда для скрытия окна Feed Reader*

Таблица 3.1. Высокоуровневое сравнение моделей программирования Warcraft и .NET

Категория	Надстройки над Warcraft	.NET
Файлы и ресурсы приложения	Оглавление (.toc)	Файл манифеста сборки
Метаданные приложения (имя, автор, номер версии)	Оглавление	AssemblyInfo
Пользовательский интерфейс	Язык разметки FrameXML и виджеты	Язык разметки XAML и элементы управления
Язык программирования	Lua	VB/C#
Модель программирования	Событийно-ориентированная	Событийно-ориентированная
Библиотеки	WoW API, сторонние библиотеки	.NET Framework, сторонние библиотеки
Хранилище	SavedVariables	Изолированное хранилище / файловая система
Ссылка на текущий объект	Self	Me (VB); this (C#)
Инструменты	AddOn Studio для *World of Warcraft*	Редакции Visual Studio Express

Краткий обзор языка программирования Lua

У языка Lua много нюансов; ниже перечислены лишь его основные особенности. Полное справочное руководство по языку Lua можно скачать со страницы *http://www.lua.org/manual/*.

Примечание

В игре *World of Warcraft* используется слегка модифицированная версия исполняющей среды Lua 5.1, в которую добавлены некоторые вспомогательные функции и исключены функции для обращения к операционной системе и файлового ввода/вывода.

Типы данных в Lua

В языке Lua определено пять базовых типов: Boolean (true/false), число, строка, таблица (словарь пар ключ/значение с дополнительными функциями) и nil (null). Как и во многих других сценарных языках, например JavaScript, переменные в Lua не являются строго типизированными, то есть при создании переменной ее тип не указывается. Не будет ошибкой впоследствии присвоить переменной значение другого типа.

Области видимости

В Lua есть две области видимости переменных: глобальная и локальная. Глобальные переменные доступны отовсюду, в том числе из других надстроек, локальные – только внутри текущей области видимости.

При разработке надстроек над *Warcraft* глобальных переменных следует по возможности избегать. Имена таких переменных должны быть уникальны, чтобы не возникало конфликтов с глобальными переменными, определенными в других надстройках.

Функции в Lua

Функции – это «кровь и плоть» Lua; весь API *World of Warcraft* основан на функциях. Во многих отношениях функции аналогичны методам .NET, но есть и несколько важных отличий, о которых следует знать:

- В Lua при вызове функции можно опускать аргументы; считается, что недостающие аргументы равны nil.
- При вызове функции можно передать больше аргументов, чем требуется; лишние аргументы игнорируются.
- Функция может принимать переменное число аргументов; это можно считать аналогом ключевого слова `ParamArray` в Visual Basic или ключевого слова `params` а C#.
- Функция в Lua может возвращать более одного значения.

В табл. 3.2 приведены основные языковые конструкции Lua.

Таблица 3.2. Краткий справочник по языку Lua

Конструкция	Код на Lua
Комментарии	`-- это комментарий`
Объявление глобальной переменной	`message = "hello"`

Объявление локальной переменной	`-- изменять тип данных разрешается`
	`local message = "hello"`
	`message = 5`

Проверка на равенство	`==`
Проверка на неравенство	`~=`

if/else	`if (message == "hello") then`
	`  print("hi!")`
	`elseif(message == "goodbye") then`
	`  print("bye!")`
	`else`
	`  print("you still here?")`
	`end`

Таблица 3.2 (продолжение)

Конструкция	Код на Lua

Конкатенация строк	`local message = "hello"`
	`local space = " "`
	`local location = "world"`
	`print(message .. space .. world)`
	`-- печатается: hello world`

Объявление функции	`function add(a,b)`
	`  return a+b`
	`end`

Объявление функции, возвращающей несколько результатов	`function returnThreeVals()`
	`return "one ", "two ", "three"`
	`end`

Вызов функции, возвращающей несколько результатов	`val1, val2, val3 = returnThreeVals()`

Цикл while	`i=1`
	`while i <= 3 do`
	`  print(i)`
	`  i = i+1`
	`end`
	`--выводится`
	`1`
	`2`
	`3`

Цикл for	`for i=1,3 do`
	`  print(i)`
	`end`
	`--выводится`
	`1`
	`2`
	`3`

Таблицы в Lua

Таблицы в Lua – это универсальная структура данных. Их можно использовать для реализации поведения перечислений, классов, массивов, наборов и т. д. В табл. 3.3 приведены некоторые типичные операции с таблицами Lua.

Примечание

В отличие от .NET, где индексация массивов и наборов начинается с нуля, в Lua таблицы индексируются с 1, то есть индекс первого элемента равен 1, а не 0.

Таблица 3.3. Краткий справочник по таблицам Lua

Конструкция	Код на Lua
Создание пустой таблицы	`feeds = {}`

Создание таблицы с одновременной инициализацией	`feeds = {`
	`{["Title"] = "Coding4Fun"},`
	`{["Title"] = "Channel 9"},`
	`{["Title"] = "C4FBook"},`
	`}`

Подсчет числа элементов в таблице	`print(#feeds)`
	`--печатается`
	`3`

Печать первого элемента	`print(feeds[1].Title)` `--печатается "Coding4Fun"`

Обход таблицы в цикле for	`-- цикл от 1 до длины таблицы`
	`for i=1,#feeds`
	`  print(feeds[i].Title)`
	`end`
	`--печатается`
	`Coding4Fun`
	`Channel 9`
	`C4FBook`

Обход таблицы в цикле foreach с использованием функции `ipairs`	`--Примечание: функция ipairs()`
	`--возвращает два результата:`
	`--индекс и значение`
	
	`for index,value in ipairs(feeds) do`
	`  print(index .. ", value=" .. value.Title);`
	`end`
	
	`-- печатается`
	`1, value=Coding4Fun`
	`2, value=Channel 9`
	`3, value=C4FBook`

Таблица Lua, используемая в Feed Reader

Утилита Feed Grabber (описана ниже в этой главе) сериализует RSS-ленты, найденные в списке Windows Common Feed List, в виде табличной переменной Lua с именем `FEED_READER_FEEDS`. Точный формат этой структуры данных показан в примере 3.1.

Пример 3.1. RSS-ленты, сериализованные в виде таблицы Lua

```
FEED_READER_FEEDS = {
  { -- начало первой ленты
```

```
    ["Title"] = "BBC News | News Front Page | UK Edition",
    ["Items"] = {
      {
        ["Title"] = "Politician caught in scandal",
        ["Content"] = "Shock, outrage and dismay",
      }, -- [1]
      {
        ["Title"] = "Coding4Fun book wins Pulitzer prize",
        ["Content"] = "Authors, media not surprised by latest award.",
      }, -- [2]
    }, -- конец списка новостей
  }, -- конец первой ленты

  { -- начало второй ленты
    ["Title"] = "NYT > NYTimes.com Home",
    ["Items"] = {
      {
        ["Title"] = "Economy worsening, analysts say",
        ["Content"] = "New data shows economy worsening.",
      }, -- [1]
    }, -- конец списка новостей
  }, -- конец второй ленты
}
```

Common Feed List – это структура на уровне операционной системы, в которой хранится список RSS-лент, на которые пользователь подписался. Наличие единого списка позволяет использовать ленты в разных приложениях, например Outlook и Internet Explorer, не подписываясь в каждом приложении заново.

Обратите внимание, что в показанном выше фрагменте таблица FEED_READER_FEEDS содержит вложенные таблицы. Продемонстрируем несколько способов обращения к этим данным из программы:

Получить количество лент

```
print("Счетчик лент: " .. #FEED_READER_FEEDS)
```

```
Счетчик лент: 2 - на верхнем уровне две ленты
```

Получить количество новостей во второй ленте

```
print("Счетчик новостей: " .. #FEED_READER_FEEDS[2].Items)
```

```
Счетчик новостей: 1 - Во второй ленте только одна новость
```

Получить текст второй новости в первой ленте

```
print(FEED_READER_FEEDS[2].Items[1].Content)
New data shows economy worsening.
```

Присвоить первую ленту другой табличной переменной

```
MY_TABLE = FEED_READER_FEEDS[1];
```

Напечатать заголовки всех новостей

```
for i=1,#FEED_READER_FEEDS do
  print(FEED_READER_FEEDS[i].Title)
end
```

```
BBC News | News Front Page | UK Edition
NYT > NYTimes.com Home
```

Язык разметки FrameXML

Пользовательский интерфейс надстройки над *World of Warcraft* определяется на языке FrameXML, который является диалектом XML. Этот язык очень похож на XAML – язык разметки, применяемый в Windows Presentation Foundation (WPF) и Silverlight. Как и XAML, язык FrameXML позволяет декларативно создавать объекты, задавая их свойства. Позже к этим объектам можно будет обратиться из программы. В языке FrameXML определен ряд элементов управления (которые в WoW называются виджетами), аналогичных тем, что имеются в WPF и Windows Forms; см. табл. 3.4.

Таблица 3.4. Виджеты FrameXML и их аналоги в .NET

FrameXML	WPF	Windows Forms
Frame	Window	Form
FontString	Label	Label
Button	Button	Button
CheckButton	CheckBox	CheckBox
Texture	Image	Image
EditBox	TextBox	TextBox

Язык FrameXML позволяет создавать сложные динамические макеты посредством якорей и шаблонов. Шаблоны FrameXML похожи на комбинацию шаблонов, стилей и нестандартных элементов управления WPF. Прежде чем приступать к их изучению, давайте рассмотрим разметку Hello World в примере 3.2.

Пример 3.2. Hello World на языке FrameXML

```
<Ui … >
  <Script file="Frame.lua" />
  <Frame name="Frame1" parent="UIParent" toplevel="true" enableMouse="true">
    <Size>
      <AbsDimension x="200" y="150" />
    </Size>
    <Anchors>
      <Anchor point="CENTER" />
```

```
    </Anchors>
…
    <Layers>
      <Layer level="OVERLAY">
        <FontString name="FontString1" inherits="GameFontNormal"
                    text="Hello world!">
          <Anchors>
            <Anchor point="CENTER">
              <Offset x="0" y="20" />
            </Anchor>
          </Anchors>
        </FontString>
      </Layer>
    </Layers>
    <Frames />
    <Scripts />
  </Frame>
</Ui>
```

В начале мы определяем ассоциированный с файлом разметки файл сценария *Frame.lua*. Аналогично выделению программной логики в отдельный файл в .NET обработчики событий могут быть вынесены в файл *Frame.lua*.

Затем объявляем фрейм `Frame1`, задавая для него некоторые свойства, в частности устанавливаем ширину (200 пикселов) и высоту (150 пикселов) фрейма. Поскольку задан якорь `CENTER`, то фрейм будет расположен в центре экрана.

Фрейм содержит виджет `FontString` с именем `FontString1`, который располагается со смещением в 20 пикселов вверх от центра фрейма.

FrameXML-разметка из примера 3.2 визуализируется внутри игры в виде пользовательского интерфейса, показанного на рис. 3.4.

***Рис. 3.4.** Фрейм Hello world!*

К любому объявленному в FrameXML-разметке элементу пользовательского интерфейса можно обратиться из Lua-сценария с помощью одноименного глобального объекта. Можете убедиться в этом, набрав в окне чата такую команду:

```
"/script HelloWorldFrameText:SetText("Hello Coding4Fun!");
```

Избегайте конфликтов имен виджетов

Хотя мы пока рассматриваем лишь основы FrameXML, важно с самого начала понимать, что все виджеты, объявленные в FrameXML-разметке надстройки, должны иметь уникальные имена, чтобы не вступать в конфликт с другими надстройками.

Один из способов избежать конфликта имен состоит в том, чтобы использовать специальный макрос *Warcraft* `$parent`, вместо которого автоматически подставляется имя родителя. В примере 3.3 мы могли бы переименовать `FontString1` в `$parentText`; поскольку родителем в данном случае является фрейм Frame1, то имя автоматически будет преобразовано в `Frame1Text`.

Верстка

У каждого виджета в FrameXML может быть размер и один или несколько якорей. Перед выводом надстройки на экран *World of Warcraft* определяет истинное положение и размеры виджетов, исходя из значений этих параметров.

Размер фрейма можно задавать абсолютно или относительно родителя.

```
<!-- Кнопка с абсолютным размером 100 x 20 пикселов -->
<Button>
  <Size>
    <AbsDimension x="100" y="20" />
  </Size>
</Button>

<!-- Фрейм, вдвое меньший своего родителя -->
<Frame>
  <Size>
    <RelDimension x="0.5" y="0.5" />
  </Size>
</Frame
```

Якорь определяет точку, в которой размещается виджет, и всегда задается относительно другого виджета или точки на экране. В *World of Warcraft* существует девять точек привязки (рис. 3.5).

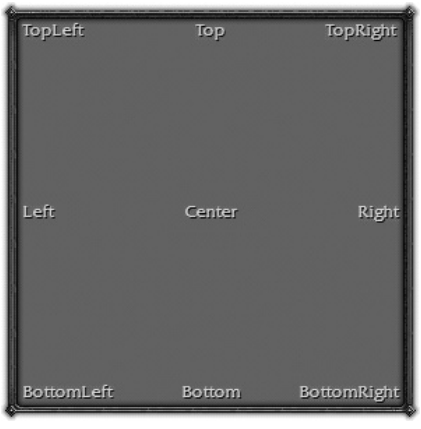

Рис. 3.5. *Девять точек привязки виджета*

В табл. 3.5 перечислены свойства якоря в языке FrameXML.

Таблица 3.5. Свойства якоря

Свойство анкера	Описание
`point`	Обязательное. Определяет точку, к которой привязывается виджет.
`relativeTo`	Необязательное. Имя виджета, относительно которого задается якорь. Если опущено, то используется родительский виджет. Если у виджета нет родителя, то используется экран.
`relativePoint`	Необязательное. Точка виджета, к которому привязан якорь (см. relativeTo). Если опущено, то используется та же точка привязки, что задана в свойстве point.
`offset`	Необязательное. Смещение в пикселах относительно точки relativePoint.

В примере 3.3 якорь позиционирует кнопку `CancelButton` рядом с `OKButton`, оставляя между ними 10 пикселов. Кнопка `OKButton` находится в правом нижнем углу своего родителя с отступом 15 пикселов.

Пример 3.3. Привязка двух элементов управления друг к другу

```
<Button name="OKButton" inherits="UIPanelButtonTemplate" text="OK">
  <Size>
    <AbsDimension x="80" y="25" />
  </Size>
  <Anchors>
    <Anchor point="BOTTOMRIGHT">
      <Offset>
```

```
        <AbsDimension x="-15" y="15" />
      </Offset>
    </Anchor>
  </Anchors>
</Button>

<Button name="CancelButton" inherits="UIPanelButtonTemplate" text="Cancel">
  <Size>
    <AbsDimension x="80" y="25" />
  </Size>
  <Anchors>
    <Anchor point="RIGHT" relativePoint="LEFT" relativeTo="OKButton">
      <Offset>
        <AbsDimension x="-10" y="0" />
      </Offset>
    </Anchor>
  </Anchors>
</Button>
```

Конструирование FrameXML в AddOn Studio

Те разработчики программ для .NET, которые жизни не мыслят без Visual Studio, могут воспользоваться бесплатным инструментом с открытыми исходными текстами AddOn Studio (AOS) для *World of Warcraft*, который служит аналогом Visual Studio в мире создания надстроек для *Warcraft*. AOS обладает рядом функций, хорошо знакомых разработчикам для .NET, в том числе визуальным конструктором, инструментарием элементов управления, механизмом автоматической сборки файлов развертывания (TOC), IntelliSense, окном событий и свойств, средством выполнения рефакторинга, механизмом решений и т. д. На рис. 3.6 показано, как в AddOn Studio выглядит FrameXML-файл из примера 3.3.

Вместо того чтобы вручную задавать свойства якорей в FrameXML, вы можете выбрать кнопку Cancel и щелкнуть по свойству Anchors в окне свойств, – появится диалоговое окно Anchor Collection Editor, показанное на рис. 3.7. Этот редактор коллекции якорей позволяет визуально определить все свойства FrameXML для кнопки Cancel из примера 3.3.

Шаблоны

Подобно стилям в WPF, шаблоны позволяют задать единый внешний облик виджетов FrameXML. В *World of Warcraft* определен ряд шаблонов стандартных элементов пользовательского интерфейса, которыми по мере необходимости удобно пользоваться.

Чтобы воспользоваться шаблоном виджета, например `FontString`, нужно лишь записать в атрибут `inherits` имя существующего шаблона `FontString`. В AddOn Studio вы можете унаследовать любой из встроенных шаблонов, задав значение свойства `inherits` в окне свойств, как показано на рис. 3.8.

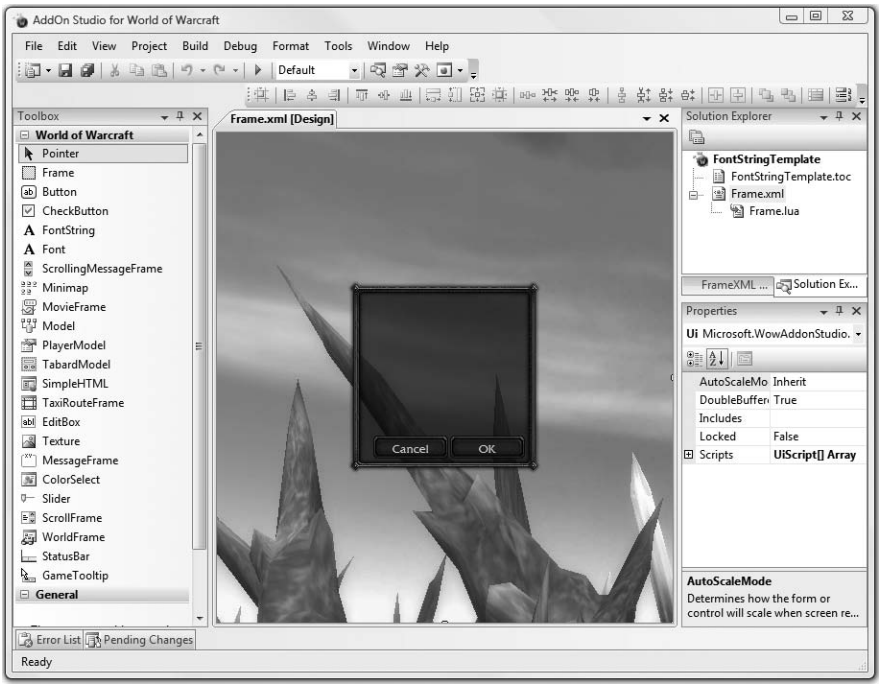

Рис. 3.6. *FrameXML-файл из примера 3.3 в AddOn Studio*

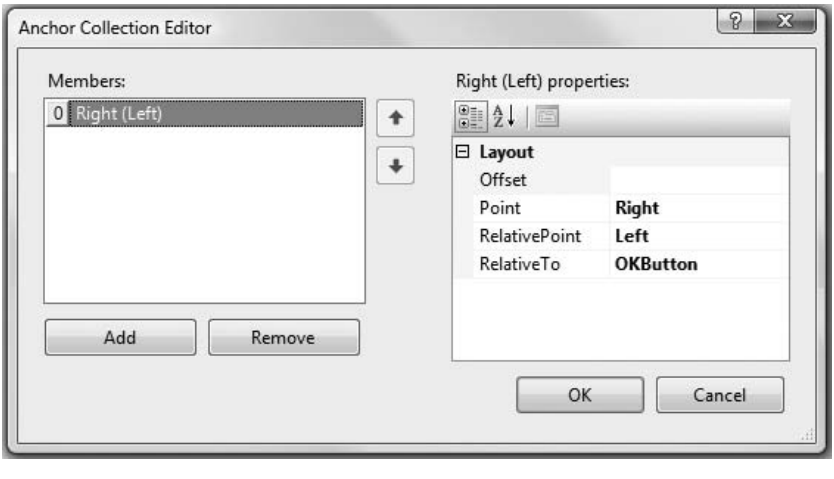

Рис. 3.7. *Окно Anchor Collection Editor*

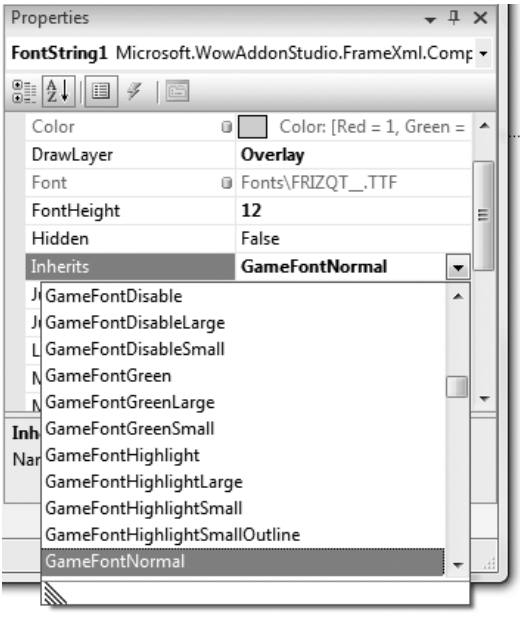

Рис. 3.8. *Предопределенные в Warcraft шаблоны для FontString*

События

Подобно элементам управления .NET, для виджетов FrameXML определены события, с которыми можно ассоциировать функции-обработчики примерно так же, как мы назначаем обработчик события щелчка по кнопке в приложении для .NET. Lua-функция может быть определена прямо в FrameXML-файле или связана с событием, то есть функция будет вызываться при возникновении этого события. В примере 3.4 приведен встроенный обработчик события `OnClick` кнопки.

Пример 3.4. Обработка события нажатия кнопки OnClick

```
<Button name="ChattyButton" inherits="UIPanelButonTemplate" text="Click me!">
  <Anchors>
    <!-- ... -->
  </Anchors>
  <Scripts>
    <OnClick>
      message("Hey there!!");
    </OnClick>
  </Scripts>
</Button>
```

Одним из самых важных является событие `OnLoad`, которое возникает после загрузки фрейма. В AddOn Studio обработчики события можно

сгенерировать так же, как в проектах WPF или Windows Forms, – достаточно щелкнуть по имени события на вкладке событий в окне свойств (рис. 3.9).

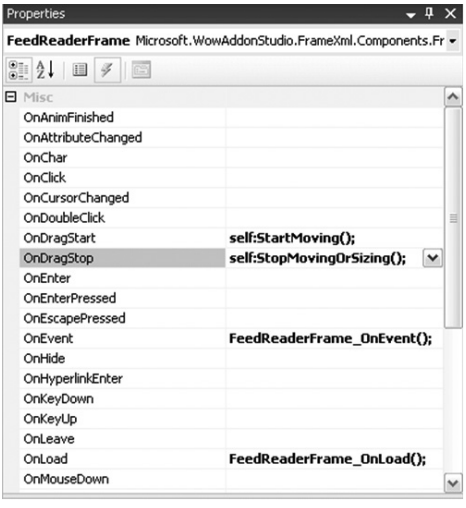

Рис. 3.9. *События фрейма в AddOn Studio для World of Warcraft*

Надстройка Feed Reader

Разобравшись с принципами работы надстроек, займемся разработкой нашей надстройки Feed Reader.

Анатомия Feed Reader

Первым делом посмотрим, из каких файлов состоит надстройка Feed Reader (рис. 3.10).

Сразу же отметим, что Feed Reader находится в каталоге *Users\Public\Games\World of Warcraft\Interface\AddOns\FeedReader*. Каждая надстройка над *Warcraft* должна находиться в отдельном подкаталоге этого каталога.

В состав Feed Reader входят следующие файлы:

FeedReader.toc

Оглавление (table of contents – TOC) надстройки, в котором определены ее имя, автор, номер версии и – самое главное – список всех входящих в нее файлов. AddOn Studio создает TOC-файл автоматически, поэтому редактировать его вручную не следует (выберите пункт меню Project, если хотите посмотреть, что хранится в TOC).

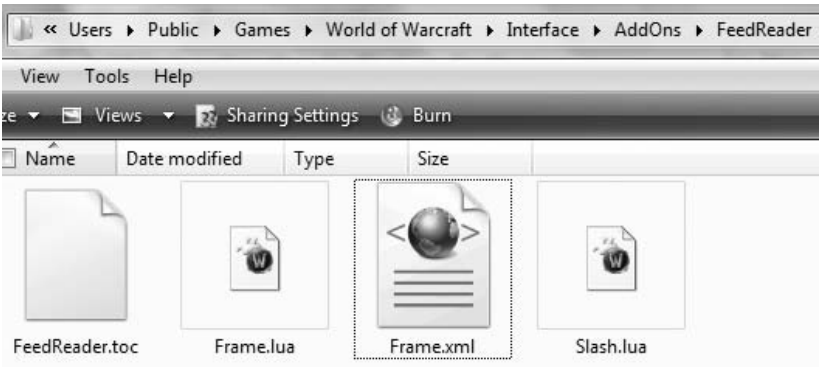

Рис. 3.10. *Четыре файла, составляющих надстройку Feed Reader*

Frame.xml

Здесь определяется внешний облик Feed Reader: кнопки, управляющие элементы, положение, цвета и т. д.

Frame.lua

Здесь определяются Lua-функции, обрабатывающие события, возбуждаемые виджетами, определенными в файле *Frame.xml*.

Slash.lua

Отдельный Lua-файл, в котором определены /-команды, то есть текстовые команды, которые можно выполнять из окна чата *Warcraft* (рис. 3.3).

Как работают Feed Reader и Feed Grabber

Взглянем на порядок взаимодействия Feed Reader и Feed Grabber с высоты птичьего полета.

1. Еще до запуска *Warcraft* уже работает в фоновом режиме программа Feed Grabber, которая отбирает новости из RSS-лент, зарегистрированных в общем списке Common Feed List (см. рис. 3.1).
2. Feed Grabber считывает RSS-ленты в память, очищает их от мусора, экранирует недопустимые символы, после чего сериализует новости в виде табличной структуры данных Lua и записывает в файл *FeedReader.lua*. Этот файл находится в каталоге *Users\Public\Games\World of Warcraft\WTF\Account\AccountName\SavedVariables*.
3. При запуске игра *World of Warcraft* просматривает все подкаталоги каталога *Users\Public\Games\World of Warcraft\Interface*. Там она найдет подкаталог надстройки Feed Reader, прочитает файл *FeedReader.TOC* и все файлы, упомянутые в разделе «Анатомия Feed Reader».

4. Feed Reader с помощью функций *Warcraft* API регистрирует событие `ADDON_LOADED` (см. примеры 3.9 и 3.10).
5. *Warcraft* видит, что в файле *FeedReader.TOC* объявлена сохраненная переменная с именем `FEED_READER_FEEDS`, загружает файл *FeedReader.lua*, сформированный на шаге 2, и присваивает переменной `FEED_READER_FEEDS` таблицу, в которой хранятся сериализованные новости.
6. *Warcraft* завершает загрузку надстройки и вызывает определенную в Feed Reader функцию `OnEvent()`, передавая ей ряд параметров, в том числе имя события и имя надстройки, возбудившей это событие (см. пример 3.11).
7. Feed Reader вызывает функцию `SelectFeed`, передавая в качестве параметра 1, то есть требуя выбрать первую ленту (пример 3.12).
8. Функция `SelectFeed` устанавливает переменную `CURRENT_FEED`, так чтобы она указывала на первую ленту в таблице `FEED_READER_FEEDS` (пример 3.12).
9. Вызывается функция `UpdateFeeds`, которая обновляет пользовательский интерфейс фрейма `FeedsScrollFrame` (пример 3.14).
10. Функция `SelectFeedItem` устанавливает переменную `CURRENT_ITEM`, так чтобы она указывала на первую новость в текущей ленте (пример 3.13).
11. Вызывается функция `UpdateFeedItems`, которая обновляет пользовательский интерфейс фрейма `FeedItemsScrollFrame` (пример 3.15).
12. Вызывается функция `UpdateSummary`, которая выводит содержимое текущей новости (пример 3.18).
13. Чтобы получить актуальные новости, игрок должен перезагрузить свой пользовательский интерфейс. Это заставит *Warcraft* записать сохраненные переменные в файл *FeedReader.lua*, упомянутый на шаге 5.
14. .NET-приложение Feed Grabber с помощью класса `FileSystemWatcher` обнаруживает, что *Warcraft* обновила файл сохраненных переменных на шаге 13, после чего заново считывает RSS-ленты и записывает их в файл *FeedReader.lua* (примеры 3.30 и 3.31).

Пользовательский интерфейс Feed Reader

Надстройка Feed Reader состоит из простого фрейма верхнего уровня с фоновой картинкой и стилями, которые AddOn Studio задала по умолчанию, и трех вложенных фреймов: фрейма RSS-лент, фрейма новостей и сводного фрейма.

В левом верхнем углу фрейма Feed Reader находится надпись `FontString`, а в правом верхнем углу – кнопка `Close`.

В примере 3.5 приведено определение фрейма верхнего уровня `FeedReaderFrame`.

Пример 3.5. Главный фрейм Feed Reader (дочерние фреймы опущены)

```
<Frame name="FeedReaderFrame" parent="UIParent" toplevel="true" movable="true"
enableMouse="true">
  <Size>
    <AbsDimension x="800" y="300" />
  </Size>
  <Anchors>
    <Anchor point="CENTER" />
  </Anchors>
  <Backdrop bgFile="Interface\DialogFrame\UI-DialogBox-Background"
    edgeFile="Interface\DialogFrame\UI-DialogBox-Border" tile="true">
    <BackgroundInsets>
      <AbsInset left="11" right="12" top="12" bottom="11" />
    </BackgroundInsets>
    <TileSize>
      <AbsValue val="32" />
    </TileSize>
    <EdgeSize>
      <AbsValue val="32" />
    </EdgeSize>
  </Backdrop>
  <Layers ... />
  <Frames ... />
  <Scripts>
    <OnDragStart>self:StartMoving();</OnDragStart>
    <OnDragStop>self:StopMovingOrSizing();</OnDragStop>
    <OnLoad>self:RegisterForDrag("LeftButton"); FeedReaderFrame_OnLoad();
    </OnLoad>
  <OnEvent>FeedReaderFrame_OnEvent();</OnEvent>
  </Scripts>
</Frame>
```

Объявлен также шаблон кнопки Button для показа лент и новостей. В этом шаблоне заданы ширина и высота новости, а также шрифты (пример 3.6).

Пример 3.6. Шаблон ScrollItemTemplate, используемый в Feed Reader

```
<Button name="ScrollItemTemplate" virtual="true">
  <Size>
    <AbsDimension x="150" y="16" />
  </Size>
  <NormalFont style="GameFontNormal" justifyH="LEFT" />
  <HighlightFont style="GameFontHighlight" justifyH="LEFT" />
</Button>
```

Фреймы

Во фреймах `FeedFrame` и `FeedItemsFrame` отображаются прокручиваемые списки. В обоих используется виджет `ScrollFrame` и пять кнопок, основанных на шаблоне `ScrollItemTemplate`. Унаследовав `ScrollFrame` от встроенного шаблона `FauxScrollFrameTemplate`, мы поручили *World of*

Warcraft заботу об отображении полосы прокрутки (хотя код прокрутки фрейма нам придется написать самостоятельно, см. раздел «Функции UpdateFeeds и UpdateFeedItems» ниже). На рис. 3.11 показано дерево всех элементов управления в файле *Frame.xml*.

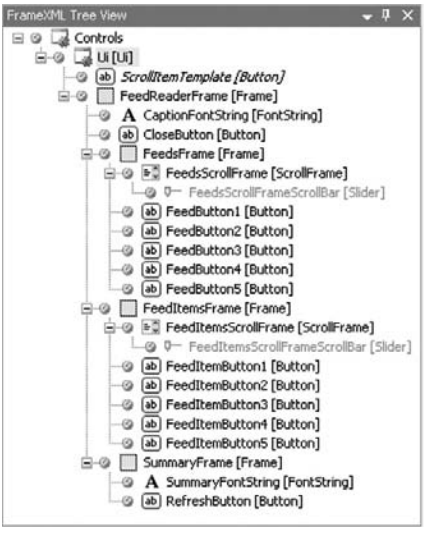

***Рис. 3.11.** Дерево пользовательского интерфейса Feed Reader*

Каждая кнопка во фреймах `FeedsFrame` и `FeedItemsFrame` содержит короткий сценарий для обработки события `OnClick`, из которого вызывается Lua-функция обновления интерфейса (см. примеры 3.18 и 3.20). В качестве параметра ей передается индекс кнопки. Каждая кнопка, кроме верхней, позиционируется путем привязки к предыдущей, а верхняя привязывается к тому фрейму, в котором находится. Разметка кнопки `FeedButton2` показана в примере 3.7.

Пример 3.7. FeedButton2

```
<Button name="FeedButton2" inherits="ScrollItemTemplate" text="Feed2">
  <Anchors>
    <Anchor point="TOPLEFT" relativeTo="FeedButton1"
            relativePoint="BOTTOMLEFT" />
    <Anchor point="TOPRIGHT" relativeTo="FeedButton1"
            relativePoint="BOTTOMRIGHT" />
  </Anchors>
  <Scripts>
    <OnClick>
      FeedButton_Clicked(2);
    </OnClick>
  </Scripts>
</Button>
```

Добавление кода для Frame.xml

Описав внешний облик Feed Reader на языке FrameXML, мы должны реализовать код, который стоит за обработкой событий щелчка мышью и отображением RSS-лент. Это код находится в файле *Frame.lua*.

Глобальные переменные

В надстройке Feed Reader определены три глобальные переменные, используемые для хранения таблиц (пример 3.8).

Пример 3.8. Три глобальные табличные переменные

```
FEED_READER_FEEDS = {}
CURRENT_FEED = {}
CURRENT_ITEM = {}
```

`FEED_READER_FEEDS`

Эта табличная переменная определена как *сохраняемая* в файле *FeedReader.TOC*. Это означает, что *Warcraft* запишет в нее сериализованное значение, хранящееся в файле *Users\Public\Games\World of Warcraft\WTF\Account\AccountName\SavedVariables* (см. пример 3.1 выше).

`CURRENT_FEED`

Здесь хранится текущая лента, когда мы в цикле обходим таблицу `FEED_READER_FEEDS`.

`CURRENT_ITEM`

Здесь хранится текущая новость, когда мы в цикле перебираем новости в ленте `CURRENT_FEED`.

Определение переменной как сохраняемой

Чтобы определить переменную как сохраняемую (то есть сохраняющую значение между запусками игры), выберите из меню AddOn Studio пункт Project→Project Properties и введите имена необходимых вам сохраняемых переменных в строке SavedVariables (рис. 3.12). Если переменных несколько, перечислите их через запятую.

Регистрация событий Warcraft

При первом запуске Feed Reader WoW смотрит, какие сценарии и события указаны в файле *Frame.xml* (пример 3.9).

Пример 3.9. Сценарии Feed Reader

```
<Scripts>
...
  <OnLoad>FeedReaderFrame_OnLoad();</OnLoad>
  <OnEvent>FeedReaderFrame_OnEvent();</OnEvent>
</Scripts>
```

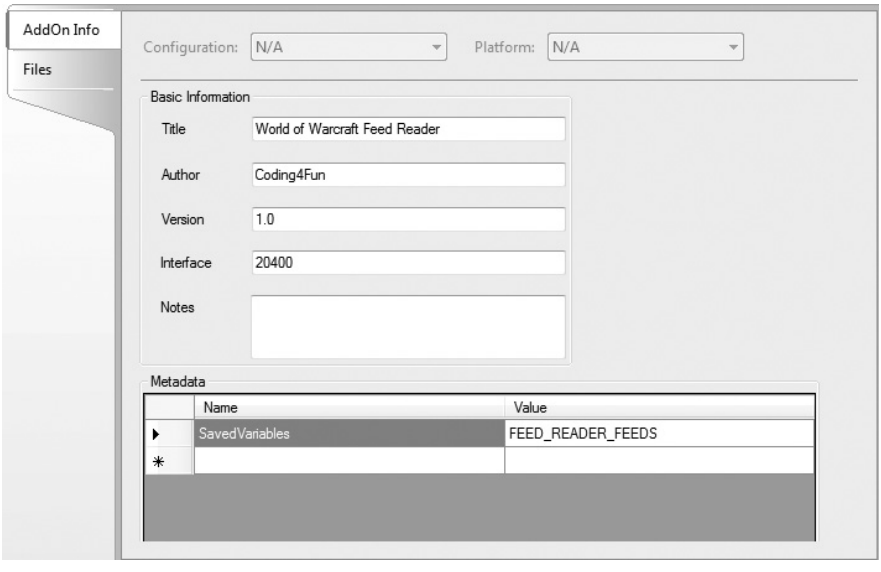

***Рис. 3.12.** Определение сохраняемой переменной в свойствах проекта AddOn Studio*

Сначала вызывается функция `OnLoad`, которая регистрирует для надстройки событие `ADDON_LOADED`, как показано в примере 3.10.

Пример 3.10. Событие загрузки Feed Reader

```
function FeedReaderFrame_OnLoad()
  this:RegisterEvent("ADDON_LOADED");
end
```

`OnEvent` — стандартная функция, которая является обработчиком всех событий *Warcraft*. Это означает, что, если вы подписываетесь на какие-нибудь другие события FrameXML или такие события игры, как запись в протокол сражений или событие чата, то все они будут передаваться функции `OnEvent`. Чтобы определить, какое событие привело к вызову `OnEvent`, можно воспользоваться глобальной переменной `event`.

С помощью переменной `event` мы можем узнать, что произошло событие `ADDON_LOADED`, и обработать его.

Вторая «фишка» состоит в том, что событие `ADDON_LOADED` возбуждается для всех установленных надстроек. Однако мы можем опросить переменную `arg1`, в которой находится имя загруженной надстройки, и вызывать наш обработчик, только если загрузилась надстройка Feed Reader.

Если словесное объяснение показалось вам непонятным, то код в примере 3.11 не оставит сомнений в том, что мы проверяем тип события и имя надстройки перед тем, как вызвать функцию `SelectFeed`.

Пример 3.11. Обработка событий внутри надстройки

```
function FeedReaderFrame_OnEvent()
  if ((event == "ADDON_LOADED") and (arg1 == 'FeedReader')) then
    SelectFeed(1);
  end
end
```

Почему функция OnLoad не может просто вызвать функцию SelectFeed?

Возможно, у вас возник вопрос, почему функция OnLoad регистрирует событие ADDON_LOADED вместо того, чтобы вызвать SelectFeed напрямую. Причина в том, что в момент вызова OnLoad сохраняемая переменная FEED_READER_FEEDS еще не загружена в память. А в момент возбуждения события ADDON_LOADED все сохраняемые переменные гарантированно загружены и доступны надстройке.

Функции SelectFeed и SelectFeedItems

Функции SelectFeed и SelectFeedItems необходимы для присваивания значений переменным CURRENT_FEED и CURRENT_ITEM соответственно.

Функции SelectFeed передается параметр feedIndex – индекс той записи таблицы FEED_READER_FEEDS, которую нужно присвоить таблице CURRENT_FEED. Она смотрит, выбрана ли уже какая-то лента, и, если нет, устанавливает переменную CURRENT_FEED. Затем вызывается функция UpdateFeeds, которая обновляет пользовательский интерфейс фрейма и обращается к функции SelectFeedItems, передавая ей параметр 1, чтобы та выбрала первую новость в ленте (пример 3.12).

Пример 3.12. Обновление текущей ленты в функции SelectFeed

```
function SelectFeed(feedIndex)
  if (FEED_READER_FEEDS[feedIndex] ~= CURRENT_FEED) then
    --Записать в CURRENT_FEED текущую ленту
    CURRENT_FEED = FEED_READER_FEEDS[feedIndex];
    UpdateFeeds();
    SelectFeedItem(1);
  end
end
```

Функция SelectFeedItem

Функция SelectFeedItem очень похожа на функцию SelectedFeed из примера 3.13, только находится на один уровень ниже. Поэтому если SelectFeed записывает в переменную CURRENT_FEED текущую ленту, то SelectFeedItem записывает в CURRENT_ITEM текущую новость. При первом

вызове этой функции мы сравниваем значение CURRENT_ITEM с новостью, хранящейся в записи CURRENT_FEED.Items с указанным индексом. Если они не совпадают, то необходимо установить значение CURRENT_ITEM, после чего вызвать функцию UpdateFeedItems, которая обновит список новостей, видимых во фрейме FeedItemsScrollFrame (пример 3.13).

Пример 3.13. Обновление текущей новости в функции SelectFeedItem

```
function SelectFeedItem(feedItemIndex)
  if (CURRENT_FEED.Items[feedItemIndex] ~= CURRENT_ITEM) then
    --Записать в CURRENT_ITEM текущую новость
    CURRENT_ITEM = CURRENT_FEED.Items[feedItemIndex];
    UpdateFeedItems();
    UpdateSummary();
  end
end
```

Функции UpdateFeeds и UpdateFeedItems

Функции UpdateFeeds и UpdateFeedItems применяются для обновления фреймов FeedsScrollFrame и FeedItemsScrollFrame соответственно.

Как разработчик на платформе .NET, вы, конечно, знакомы с такими элементами управления, как ComboBox или ListBox, которые инкапсулируют поведение, связанное с прокруткой списка. Но, поскольку в *Warcraft* таких элементов нет, то придется вручную устанавливать текст, видимый в каждом из прокручиваемых фреймов.

В каждом из фреймов FeedsScrollFrame и FeedItemsScrollFrame есть пять кнопок, которые нужно обойти в цикле и для каждой установить текстовые значения, исходя из текущей позиции в списке.

Чтобы пояснить, как это все работает, рассмотрим табл. 3.6, в которой показано представление в памяти переменной FEED_READER_FEEDS и пяти кнопок во фрейме FeedsScrollFrame.

Таблица 3.6. Установка значений, видимых во фрейме FeedsScrollFrame

Индекс	Заголовок	Кнопка
1	Feed Title 1	FeedButton1
2	Feed Title 2	FeedButton2
3	Feed Title 3	FeedButton3
4	Feed Title 4	FeedButton4
5	Feed Title 5	FeedButton5
6	Feed Title 6	Скрыта
7	Feed Title 7	Скрыта

При первом обращении функция UpdateFeeds обходит в цикле первые пять элементов таблицы и записывает текст заголовка первой новости в кнопку FeedButton1, текст заголовка второй новости – в кнопку FeedButton2 и т. д. для всех пяти кнопок.

Когда пользователь щелкает по стрелочке *прокрутки вниз* во фрейме FeedsScrollFrame, функция UpdateFeeds вызывается снова, чтобы скопировать заголовки в текст каждой кнопки. Отличие от предыдущего случая состоит в том, что мы сдвинулись на один элемент вниз, поэтому:

- первая новость больше не видна;
- вторая новость становится первой видимой, и ее заголовок записывается в текст кнопки FeedButton1;
- шестая новость, которая раньше была скрыта, теперь видима и помещается в кнопку FeedButton5.

Новое распределение значений показано в табл. 3.7.

Таблица 3.7. После прокрутки вниз видимое содержимое сдвигается на единицу

Индекс	Заголовок	Кнопка
1	Feed Title 1	Скрыта
2	Feed Title 2	FeedButton1
3	Feed Title 3	FeedButton2
4	Feed Title 4	FeedButton3
5	Feed Title 5	FeedButton4
6	Feed Title 6	FeedButton5
7	Feed Title 7	Скрыта

Функция UpdateFeeds

Теперь разберем функцию UpdateFeeds из примера 3.14, которая обновляет пользовательский интерфейс надстройки Feed Reader. Так как фрейм FeedScrollFrame наследует шаблону FauxScrollFrameTemplate, необходимо сначала сообщить фрейму, как себя вести, вызвав функцию FauxScrollFrame_Update со следующими параметрами:

- подлежащий обновлению фрейм, FeedsScrollFrame;
- общее число имеющихся лент (в нашем примере семь);
- количество одновременно видимых новостей, в данном случае пять (во фрейме FeedsScrollFrame всего пять кнопок);
- высота каждой кнопки – именно на столько пикселов мы должны прокручивать фрейм. По умолчанию высота равна 16 пикселов.

Далее нужно в цикле присвоить текст каждой из пяти кнопок. Мы должны правильно выбрать индекс начального элемента в цикле, поэтому всякий раз, как пользователь прокручивает фрейм вниз, мы увеличиваем `feedIndex` на величину, получаемую от функции `GetOffset`.

Предположим, что мы выполняем первую итерацию цикла `for` и пользователь прокрутил список один раз. Функция `GetOffset` в этом случае вернет 1, так что после сложения со счетчиком цикла `i` переменная `feedIndex` станет равна 2. Так как `feedIndex` равно 2, то в первую кнопку записывается текст второй новости (значение `feedIndex`) и так далее.

Функция `getglobal` возвращает виджет `FeedButton`, и мы записываем в текст кнопки заголовок новости с уже известным индексом. Если индекс меньше или равен числу элементов, то элемент управления делается невидимым, поскольку в нем нечего отображать.

Чтобы показать, какая лента сейчас является текущей, мы вызываем функцию `LockHighlight` (пример 3.14), которая визуально выделяет виджет.

Пример 3.14. Обновление пользовательского интерфейса FeedsScrollFrame

```
function UpdateFeeds()
  -- Передается подлежащий обновлению фрейм, общее
  -- число лент, число кнопок и высота каждой кнопки
  FauxScrollFrame_Update(FeedsScrollFrame, #FEED_READER_FEEDS, 5, 16);

  local i;
  local feedIndex;

  for i = 1, 5 do
    feedIndex = i + FauxScrollFrame_GetOffset(FeedsScrollFrame);

    if (feedIndex <= #FEED_READER_FEEDS) then
      getglobal("FeedButton" .. i):SetText(FEED_READER_FEEDS
                                           [feedIndex].Title);
      getglobal("FeedButton" .. i):Show();
    else
      getglobal("FeedButton" .. i):Hide();
    end

    if (FEED_READER_FEEDS[feedIndex] == CURRENT_FEED) then
      getglobal("FeedButton" .. i):LockHighlight();
    else
      getglobal("FeedButton" .. i):UnlockHighlight();
    end
  end
end
```

Функция UpdateFeedItems

Функция `UpdateFeedItems` работает так же, как `UpdateFeed`, с тем отличием, что обновляет фрейм `FeedItemsScrollFrame` (а не `FeedsScrollFrame`) и ус-

танавливает свойства кнопок FeedItemButton (а не FeedButton). См. пример 3.15.

Пример 3.15. Обновление пользовательского интерфейса FeedItemsScrollFrame

```
function UpdateFeedItems()
  FauxScrollFrame_Update(FeedItemsScrollFrame, #(CURRENT_FEED.Items), 5, 16);
  local i;
  local feedItemIndex;
  for i = 1, 5 do
    feedItemIndex = i + FauxScrollFrame_GetOffset(FeedItemsScrollFrame);
    if (feedItemIndex <= #(CURRENT_FEED.Items)) then
      getglobal("FeedItemButton" .. i):SetText(
         CURRENT_FEED.Items[feedItemIndex].Title);
      getglobal("FeedItemButton" .. i):Show();
    else
      getglobal("FeedItemButton" .. i):Hide();
    end
    if (CURRENT_FEED.Items[feedItemIndex] == CURRENT_ITEM) then
      getglobal("FeedItemButton" .. i):LockHighlight();
    else
      getglobal("FeedItemButton" .. i):UnlockHighlight();
    end
  end
end
```

Обновление сводного фрейма

После того как функция UpdateFeedItems завершит работу, функция SelectFeedItem вызывает UpdateSummary для обновления содержимого текущей новости (пример 3.16).

Пример 3.16. Обновление SummaryFontString

```
function UpdateSummary()
  SummaryFontString:SetText(CURRENT_ITEM.Content);
end
```

Событие OnClick для кнопок Feed

Выше при определении FrameXML-разметки для Feed Reader мы уже видели, что все кнопки во фрейме FeedsScrollFrame имеют один и тот же обработчик события OnClick. Разница только в том, что ему передаются различные индексы кнопок (FeedButton1 передает 1, FeedButton1 – 2 и т. д.). Вызванная функция (пример 3.17) получает номер кнопки, к которому добавляет смещение, чтобы узнать, какая именно новость считывается. Например:

1. Мы нажимаем кнопку FeedButton1, в которой отображается вторая новость.
2. Вызывается обработчик FeedButton_Clicked, которому передается 1.

3. Он обращается к функции GetOffset и получает от нее смещение 1, означающее, что пользователь прокрутил список вниз и теперь в первой позиции отображается вторая новость.
4. Мы передаем функции SelectFeed значение 2.

Пример 3.17. Функция OnClick для всех кнопок FeedButton

```
function FeedButton_Clicked(i)
  local feedIndex = i + FauxScrollFrame_GetOffset(FeedsScrollFrame);
  SelectFeed(feedIndex);
end
```

Событие OnClick для кнопок FeedItem

Как и в случае фрейма FeedsScrollFrame, все кнопки FeedItemButton во фрейме FeedItemsScrollFrame вызывают один и тот же обработчик события OnClick, передавая ему номер кнопки.

Мы прибавляем номер кнопки к значению, возвращенному функцией GetOffset, чтобы вычислить индекс feedItemIndex текущей новости, и вызываем SelectFeedItem, передавая ей этот индекс (пример 3.18).

Пример 3.18. Функция OnClick для всех кнопок FeedItemButton

```
function FeedItemButton_Clicked(i)
  local feedItemIndex = i + FauxScrollFrame_GetOffset(FeedItemsScrollFrame);
  SelectFeedItem(feedItemIndex);
end
```

Добавление простой /-команды

Последнее, что нужно сделать в Feed Reader, – добавить обработчик /-команд. Мы определим две команды: одна будет «показывать» фрейм FeedReaderFrame, другая – «скрывать» его. Для регистрации /-команды нужно выполнить следующие действия:

1. Написать функцию, которая будет вызываться после ввода /-команды, например FeedReader_OnCommand.
2. Завести переменную с именем, составленным из заглавных букв по следующему образцу: SLASH_ADDONNAME1. В случае Feed Reader переменная должна называться SLASH_FEEDREADER1. Число в конце имени переменной необходимо для правильной регистрации /-команды. Мы запишем в SLASH_FEEDREADER1 имя регистрируемой /-команды – в нашем случае это будет команда «/feeds».
3. Зарегистрировать написанную на шаге 1 функцию во внутренней таблице SlashCmdList, указав в качестве ключа имя надстройки (заглавными буквами).

В примере 3.19 приведен код регистрации и обработки /-команды. Когда пользователь вводит команду «/feeds show» в окне чата *Warcraft*, вызывается функция FeedReader_OnCommand, которой в качестве аргумен-

та `arg` передается строка «show». Она, в свою очередь, вызывает функцию `FeedReaderFrame:Show()` для показа фрейма.

Пример 3.19. /-команда для надстройки Feed Reader

```
function FeedReader_OnCommand(arg)
  if arg == "show" then
    FeedReaderFrame:Show();
  end
  if arg == "hide" then
    FeedReaderFrame:Hide();
  end
end

SLASH_FEEDREADER1 = "/feeds";
SlashCmdList["FEEDREADER"] = FeedReader_OnCommand;
```

Feed Grabber

Как мы только что видели, с помощью надстроек можно реализовать весьма интересные вещи, так что возможности улучшения интерфейса *World of Warcraft* ограничены только вашим воображением. Однако надстройки не умеют взаимодействовать с внешним миром. Вместе с языком Lua поставляются стандартные библиотеки, в которых есть, например, функции открытия и чтения файлов, а также доступа к службам операционной системы, но эти библиотеки по соображениям безопасности недоступны внутри *World of Warcraft*. Поэтому надстройка Feed Reader не может загрузить RSS-ленты самостоятельно. Ах, если бы только можно было получить эту информацию извне *World of Warcraft*...

Сохраняемые переменные дают отличный способ сохранить данные между сеансами, и хранятся они в обычном Lua-сценарии, который находится внутри инсталляционного каталога *World of Warcraft*. Хотя этот файл применяется для сохранения и загрузки переменных, перечисленных в разделе SavedVariables TOC-файла, никто не запрещает модифицировать его из другого приложения. Нужно только сохранить информацию в подходящий момент.

Feed Grabber – это простое приложение, которое «сидит» в панели задач и записывает содержимое RSS-лент верхнего уровня, зарегистрированных в списке Windows Common Feed List, в файл сохраняемых переменных, ассоциированный с Feed Reader.

Список Common Feed List

Список Common Feed List имеется в версии Windows XP SP2 с Internet Explorer 7, а также в Windows Vista. В нем хранится перечень RSS-лент, организованный в виде иерархии папок. Существует возможность периодически загружать эти ленты. Если Internet Explorer 7 об-

наруживает на текущей странице RSS-ленту, то на панели инструментов загорается оранжевый значок, означающий, что ленту можно добавить в список Feed List. Получить доступ к лентам, на которые вы подписаны, можно, выбрав из меню пункт Вид→Панели обозревателя→Веб-каналы или нажав комбинацию клавиш Ctrl+Shift+J (см. рис. 3.13).

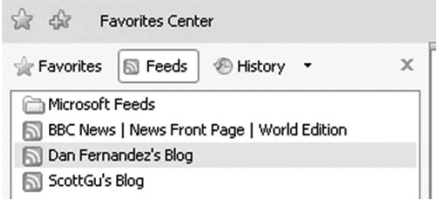

Рис. 3.13. *Список Common Feed List в Internet Explorer 7*

Для доступа к списку Common Feed List существует также API, основанный на технологии COM, а это значит, что им можно без труда воспользоваться из любого .NET-совместимого языка, например C# или VB. Чтобы добавить ссылку на этот COM-объект, нужно открыть диалоговое окно Add Reference, перейти на вкладку COM и выбрать из списка строку «Microsoft Feeds», как показано на рис. 3.14.

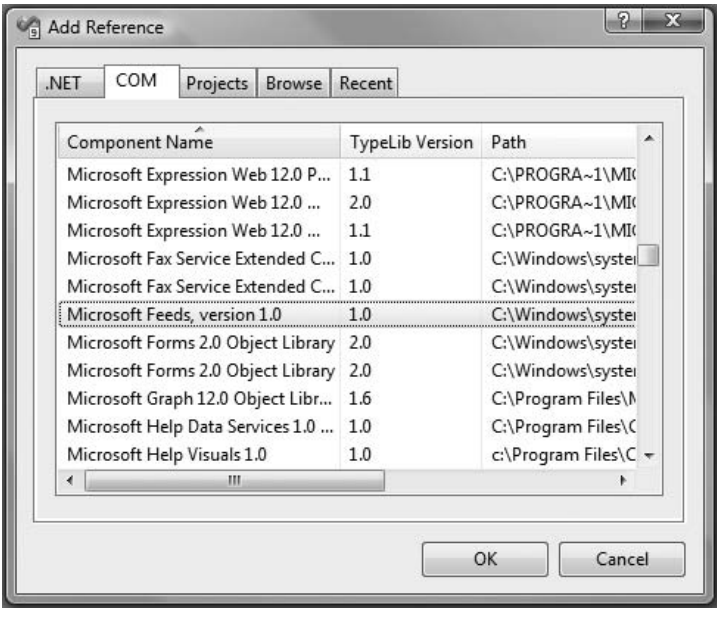

Рис. 3.14. *COM-объект Microsoft Feeds в диалоговом окне Add Reference*

Чтобы воспользоваться библиотекой Feeds, нужно сначала создать экземпляр класса FeedsManager. В этом классе есть ряд интересных нам свойств, в частности:

- иерархия всех доступных лент в свойстве RootFolder;
- признак разрешения фоновой синхронизации лент;
- принимаемый по умолчанию промежуток времени между операциями синхронизации лент.

Чтение лент из программы

Для начала определим два класса: Feed и FeedItem, в которых будем хранить RSS-ленту (см. примеры 3.20 и 3.21). Как легко заметить, эти классы повторяют структуру таблицы FEED_READER_FEEDS.

Пример 3.20. Классы Feed и FeedItem (версия на C#)

```
public class Feed
{
  public string Title { get; set; }
  public FeedItem[] Items { get; set; }
}

public class FeedItem
{
  public string Title { get; set; }
  public string Content { get; set; }
}
```

Пример 3.21. Классы Feed и FeedItem (версия на VB)

```
Public Class Feed
  Private privateTitle As String
  Public Property Title() As String
    Get
      Return privateTitle
    End Get
    Set(ByVal value As String)
      privateTitle = value
    End Set
  End Property

  Private privateItems As FeedItem()
  Public Property Items() As FeedItem()
    Get
      Return privateItems
    End Get
    Set(ByVal value As FeedItem())
      privateItems = value
    End Set
  End Property
End Class
```

```
Public Class FeedItem
  Private privateTitle As String
  Public Property Title() As String
    Get
      Return privateTitle
    End Get
    Set(ByVal value As String)
      privateTitle = value
    End Set
  End Property

  Private privateContent As String
  Public Property Content() As String
    Get
      Return privateContent
    End Get
    Set(ByVal value As String)
      privateContent = value
    End Set
  End Property
End Class
```

Определив свои классы, мы далее обратимся к COM-объектам Microsoft Feeds, чтобы извлечь RSS-ленты из списка Common Feed List. Для этого нам понадобится метод `GetTopLevelFeeds` класса `FeedGrabber`, который получает все ленты верхнего уровня из `rootFolder`, загружает те из них, которые еще не загружены, и возвращает список лент (см. примеры 3.22 и 3.23).

Пример 3.22. Возврат списка объектов Feed из Common Feed List (версия на C#)

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.IO;
using System.Threading;
using System.Lua;
using System.Lua.Serialization;
using Microsoft.Feeds.Interop;
using System.Diagnostics;

public class FeedGrabber
{
  private IFeedsManager feedsManager;
...
  private Feed[] GetTopLevelFeeds()
  {
    var rootFolder = (IFeedFolder)feedsManager.RootFolder;
    var topLevelFeeds = (IFeedsEnum)rootFolder.Feeds;

    List<Feed> feeds = new List<Feed>(topLevelFeeds.Count);
```

```
    foreach (IFeed feed in topLevelFeeds)
    {
      // Если лента еще не загружена, загрузим сейчас
      if (feed.DownloadStatus != FEEDS_DOWNLOAD_STATUS.FDS_DOWNLOADED)
        feed.Download();

      // Если ленту не удается загрузить, пропускаем ее
      if (feed.DownloadStatus != FEEDS_DOWNLOAD_STATUS.FDS_DOWNLOADED)
        continue;
      feeds.Add(CreateFeed(feed));
    }
    return feeds.ToArray();
  }
```

Пример 3.23. Возврат списка объектов Feed из Common Feed List (версия на VB)

```
Imports Microsoft.VisualBasic
Imports System
Imports System.Collections.Generic
Imports System.Linq
Imports System.IO
Imports System.Threading
Imports System.Lua
Imports System.Lua.Serialization
Imports Microsoft.Feeds.Interop
Imports System.Diagnostics

Public Class FeedGrabber
  Private feedsManager As IFeedsManager
...
  Private Function GetTopLevelFeeds() As Feed()
    Dim rootFolder = CType(feedsManager.RootFolder, IFeedFolder)
    Dim topLevelFeeds = CType(rootFolder.Feeds, IFeedsEnum)

    Dim feeds As List(Of Feed) = New List(Of Feed)(topLevelFeeds.Count)

    For Each feed As IFeed In topLevelFeeds
      ' Если лента еще не загружена, загрузим сейчас
      If feed.DownloadStatus <> FEEDS_DOWNLOAD_STATUS.FDS_DOWNLOADED Then
        feed.Download()
      End If

      ' Если ленту не удается загрузить, пропускаем ее
      If feed.DownloadStatus <> FEEDS_DOWNLOAD_STATUS.FDS_DOWNLOADED Then
        Continue For
      End If

      feeds.Add(CreateFeed(feed))
    Next feed

    Return feeds.ToArray()
End Function
```

Как вы, вероятно, заметили, в последней строке цикла `foreach` вызывается метод `CreateFeed`. Он преобразует объект `Microsoft.Feed.IFeed` в объект `Feed`. Метод `CreateFeed`, в свою очередь, вызывает метод `CreateFeedItem`, чтобы преобразовать объект `Microsoft.Feed.IFeedItem` в `FeedItem` (см. примеры 3.24 и 3.25). В конечном итоге все ленты и новости возвращаются в массиве объектов типа `Feed`.

Пример 3.24. Создание объектов Feed и FeedItem (версия на C#)

```
private static Feed CreateFeed(IFeed feed)
{
  var feedItems = (IFeedsEnum)feed.Items;
  return new Feed
    {
      Title = feed.Title,
      Items = feedItems.OfType<IFeedItem>().Select(feedItem =>
        CreateFeedItem(feedItem)).ToArray()
    };
  }

  private static FeedItem CreateFeedItem(IFeedItem feedItem)
  {
    return new FeedItem
    {
      Title = feedItem.Title,
      Content = feedItem.Description
    };
  }
```

Пример 3.25. Создание объектов Feed и FeedItem (версия на VB)

```
Private Shared Function CreateFeed(ByVal feed As IFeed) As Feed
  Dim feedItems = CType(feed.Items, IFeedsEnum)
  Return New Feed With {.Title = feed.Title, _
    .Items = feedItems.OfType(Of IFeedItem)().Select(Function(feedItem) _
    CreateFeedItem(feedItem)).ToArray()}
End Function

Private Shared Function CreateFeedItem(ByVal feedItem As IFeedItem)
        As FeedItem
  Return New FeedItem With {.Title = feedItem.Title,
                            .Content = feedItem.Description}
End Function
```

Сериализация в виде Lua-таблицы

Итак, у нас есть объекты `Feed` и `FeedItem`, но их надо сериализовать и записать в файл *FeedReader.Lua*, где находится сохраняемая переменная `FEED_READER_FEEDS`. Для этого мы воспользуемся сторонней библиотекой классов `System.Lua`, которые спроектированы аналогично классам в пространстве имен `System.Xml`.

`System.Lua` включает класс `LuaSerializer`, который с помощью отражения сериализует классы и массивы, представляя их в виде таблиц Lua, а простые типы и строки отображает на соответствующие типы Lua. Саму запись в файл на диске осуществляет класс `LuaWriter`, порождающий быстрые, не-кэшируемые, односторонние потоки над файлами с Lua-кодом.

Для сериализации ленты в файле *FeedReader.Lua* мы вызовем метод `SaveFeeds`, который получает на входе массив объектов `Feed`, инициализирует объект `LuaWriter` путем к файлу сохраняемых переменных *FeedReader.lua* и записывает данные в поток (см. примеры 3.26 и 3.27).

Пример 3.26. Метод SaveFeeds (версия на C#)

```
public void SaveFeeds()
{
  Feed[] feeds = this.GetTopLevelFeeds();
  if (feeds != null)
  {
    using (var luaWriter = LuaWriter.Create(savedVariablesPath))
    {
      var luaSerializer = new LuaSerializer();

      luaWriter.WriteStartAssignment(feedsVariableName);
      luaSerializer.Serialize(luaWriter, feeds);
      luaWriter.WriteEndAssignment();
    }
  }
}
```

Пример 3.27. Метод SaveFeeds (версия на VB)

```
Public Sub SaveFeeds()
  Dim feeds() As Feed = Me.GetTopLevelFeeds()
  If feeds IsNot Nothing Then
    Using Writer = LuaWriter.Create(savedVariablesPath)
      Dim luaSerializer = New LuaSerializer()

      Writer.WriteStartAssignment(feedsVariableName)
      luaSerializer.Serialize(Writer, feeds)
      Writer.WriteEndAssignment()
    End Using
  End If
End Sub
```

Класс `LuaWriter` гарантирует, что все записанные в файл сохраняемых переменных строки корректны с точки зрения языка Lua, так что Feed Reader сможет показать их внутри *World of Warcraft*.

Вставка обновлений путем мониторинга файла сохраняемых переменных

В момент загрузки *World of Warcraft* считывает данные из файла сохраняемых переменных, в нашем случае *FeedReader.lua*, и инициализирует переменную `FEED_READER_FEEDS`.

В ходе игры, когда персонаж игрока переключает зону, пользовательский интерфейс перезагружается, и вместо того, чтобы читать данные из файла *FeedReader.lua*, *World of Warcraft* переписывает данные, хранящиеся в этом файле. Это означает, что после начальной загрузки все обновления, которые внесла в файл сохраняемых переменных программа Feed Grabber (да и любая другая), будут затерты!

Но есть и хорошая новость – между записью в файл сохраняемых переменных и последующим чтением из него проходит пара секунд, и именно в эти драгоценные мгновения мы можем подсунуть игре наши данные!

Для этого Feed Grabber с помощью объекта `FileSystemWatcher` следит за изменениями в файле сохраняемых переменных, чтобы засечь момент, когда *World of Warcraft* записывает в него данные в процессе перезагрузки пользовательского интерфейса. Вот в этот момент мы и можем вставить в файл новые данные.

Первым делом инициализируем объект `FileSystemWatcher` в методе `InitializeFileSystemWatcher`, так чтобы он отслеживал события изменения и удаления файла *FeedReader.lua* (см. примеры 3.28 и 3.29).

Пример 3.28. Метод InitializeFileSystemWatcher (версия на C#)

```
private void InitializeFileSystemWatcher()
{
  FileSystemWatcher watcher = new FileSystemWatcher(
    Path.GetDirectoryName(savedVariablesPath),
         Path.GetFileName(savedVariablesPath));
  watcher.NotifyFilter = NotifyFilters.LastAccess | NotifyFilters.LastWrite |
    NotifyFilters.FileName;
  watcher.Changed += OnFileChangedOrDeleted;
  watcher.Deleted += OnFileChangedOrDeleted;
  watcher.EnableRaisingEvents = true;
}
```

Пример 3.29. Метод InitializeFileSystemWatcher (версия на VB)

```
Private Sub InitializeFileSystemWatcher()
  fileSystemWatcher = New FileSystemWatcher(Path.GetDirectoryName( _
    savedVariablesPath), Path.GetFileName(savedVariablesPath))
  fileSystemWatcher.NotifyFilter = NotifyFilters.LastAccess Or _
    NotifyFilters.LastWrite Or NotifyFilters.FileName
  AddHandler fileSystemWatcher.Changed, AddressOf OnFileChangedOrDeleted
  AddHandler fileSystemWatcher.Deleted, AddressOf OnFileChangedOrDeleted
```

```
    fileSystemWatcher.EnableRaisingEvents = True
End Sub
```

Теперь при любом изменении или удалении файла сохраняемых переменных будет вызван метод `OnFileChangedOrDeleted`. В нем мы прежде всего сбрасываем свойство `EnableRaisingEvents`. Это необходимо, потому что внутри метода мы сами изменяем файл, поэтому, не отключи мы мониторинг, тот же метод был бы вызван еще раз, и программа рекурсивно зациклилась бы.

Далее мы пытаемся сохранить новые данные, заменяя в цикле старые значения в файле *FeedReader.lua* каждые пять миллисекунд (примеры 3.30 и 3.31). Когда данные будут записаны, мы выходим из цикла и снова включаем свойство `EnableRaisingEvents`. Теперь наша надстройка загрузит в *Warcraft* обновленные RSS-ленты!

Пример 3.30. Запись данных в файл сохраняемых переменных (версия на C#)

```
private void OnFileChangedOrDeleted(object sender, FileSystemEventArgs e)
{
  // Приостановить получение событий обновления
  fileSystemWatcher.EnableRaisingEvents = false;

  bool success = false;

  do
  {
    try
    {
      this.SaveFeeds();
      success = true;
    }
    catch (IOException) { Thread.Sleep(5); }
  } while (!success);

  // Возобновить получение событий обновления
  fileSystemWatcher.EnableRaisingEvents = true;
}
```

Пример 3.31. Запись данных в файл сохраняемых переменных (версия на VB)

```
Private Sub OnFileChangedOrDeleted(ByVal sender As Object, _
  ByVal e As FileSystemEventArgs)

  ' Приостановить получение событий обновления
  fileSystemWatcher.EnableRaisingEvents = False
  Debug.WriteLine("inside FileChanged, time: " &
        DateTime.Now.TimeOfDay.ToString())
  Dim success As Boolean = False
  Do
    Try
      Me.SaveFeeds()
      success = True
```

```
    Catch e1 As IOException
      Thread.Sleep(5)
    End Try
  Loop While Not success

  ' Возобновить получение событий обновления
  fileSystemWatcher.EnableRaisingEvents = True
End Sub
```

Заключительные замечания

Хотя эта глава и не предоставляет исчерпывающего описания всего, что необходимо для написания собственных надстроек, надеемся, что представление о том, как увязаны между собой различные их компоненты, вы получили. Дополнительную информацию о программировании надстроек над *Warcraft* см. в книге «World of Warcraft Programming», опубликованной издательством Wiley Publishing. Бесплатный онлайновый источник справочной информации по Warcraft API – сайты *http://www.wowprogramming.com* и *http://www.wowwiki.com* – являются бесценным ресурсом для начинающих. Не забывайте также, что вы можете открыть любую стороннюю надстройку и посмотреть, как она устроена. Изучение исходного текста хорошо зарекомендовавших себя надстроек – отличный способ освоить наилучшие подходы к их разработке.

Конечно, чтобы как следует разобраться в написании надстроек над *World of Warcraft*, придется приложить определенные усилия, но возможность считывать и записывать данные, не выходя из игры, обогатит ваш игровой опыт целым классом новых приложений. Представьте себе, что вы можете читать и отправлять электронную почту из игры, или получать актуальную статистику противника с сайта Wowarmory.com, или проверять календарь рейдовых операций своей гильдии, или даже, не прерывая игры, обмениваться текстовыми сообщениями с друзьями, чтобы они могли присоединиться к вам. Все это возможно и реализуется гораздо проще, чем может показаться.

II

Программы для веб

4

InnerTube: скачиваем, конвертируем и синхронизируем видеоролики с YouTube

Автор	Дэн Фернандес
Сложность	Высокая
Необходимое время	20+ часов
Стоимость	бесплатно
Программное обеспечение	Visual C# 2008 Express с Service Pack 1, Visual Basic 2008 Express Edition с Service Pack 1, iTunes версии 7.0 или выше
Оборудование	Необязательно: устройство iPod, iPhone или Zune для синхронизации
Адрес в Интернете	*http://www.codeplex.com/InnerTube/* и *http://www.c4fbook.com/InnerTube*

Признаюсь, я люблю YouTube. Это единственный сайт в Интернете, где найдутся видеоматериалы на любой вкус: от фрагментов классических фильмов типа «Криминальное чтиво» или «Большой Лебовски» до таких интернет-знаменитостей, как Трон Гай (Tron Guy), Кимбо Слайс (Kimbo Slice) или Тай Зондай (Tay Zonday) (он же Mr. Chocolate Rain). Можно сидеть и смотреть бесконечно.

Короче говоря, на сайте YouTube есть все, кроме транспортабельности. Увы, смотреть видео с YouTube можно только при наличии соединения с Интернетом. Цель проекта InnerTube – снять это досадное ограничение. Мы научимся автоматически скачивать видео с YouTube,

конвертировать их в более подходящий формат и даже копировать на устройство iPod или Zune.

Краткий обзор

Чтобы понять, на что способна программа InnerTube, вкратце рассмотрим ее основные функции.

Пользовательский интерфейс InnerTube

На рис. 4.1 видно, что главное окно InnerTube состоит из трех панелей и по структуре аналогично Outlook. В левой панели отображается перечень лент, в средней – список видеороликов в выбранной ленте, а в правой – детальная информация о выбранном ролике.

Рис. 4.1. *Окно программы InnerTube*

Первый запуск InnerTube

При первом запуске InnerTube проверяет, установлено ли на вашем компьютере клиентское программное обеспечение iTunes или Zune и предлагает указать папку, в которой будут храниться скачанные ролики (услужливо предлагая вариант по умолчанию). Если ни iTunes, ни Zune не обнаружены, то функция синхронизации с ними отключается (см. рис. 4.2).

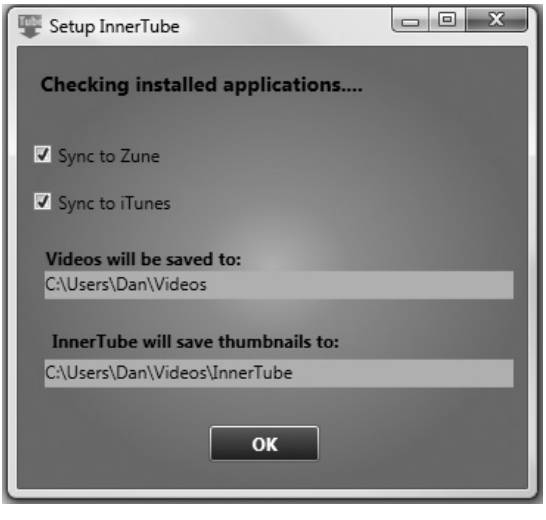

Рис. 4.2. *InnerTube конфигурирует себя для работы на вашем ПК*

Добавление видео

Выбрав из главного меню пункт Add Videos, вы сможете без труда добавить различные типы лент с YouTube (рис. 4.3). Для каждой ленты можно задать различные свойства, например: имя пользователя, избранные ролики которого вас интересуют, или интервал времени между скачиваниями роликов с наивысшим рейтингом (Top Rated Over Time).

Как работает InnerTube

Познакомившись с тем, что программа InnerTube делает, рассмотрим, как она устроена, а именно:

- принцип работы InnerTube API;
- классы InnerTube для работы с роликами и лентами;

Рис. 4.3. *Добавление лент в InnerTube*

- преобразование XML-документов, возвращаемых YouTube API, в объекты InnerTube;
- порядок загрузки видеороликов с YouTube;
- конвертирование видеоформатов;
- краткий обзор фоновых процессов InnerTube;
- синхронизация с iTunes и Zune;
- построение приложения InnerTube;
- конструирование пользовательского интерфейса InnerTube, основанного на технологии WPF.

Принцип работы YouTube API

YouTube API представляет собой REST-службу, в которой используется формат Google Gdata, позволяющий выполнять такие операции, как поиск и извлечение информации о видеороликах, а также загрузку роликов на сайт YouTube.

В отличие от других служб, например Live.com или Amazon.com, не требуется заводить учетную запись разработчика и получать ключ API, чтобы читать данные с YouTube. На рис. 4.4 показаны результаты запроса роликов с самым высоким рейтингом за все время существования YouTube, полученные с помощью броузера Firefox.

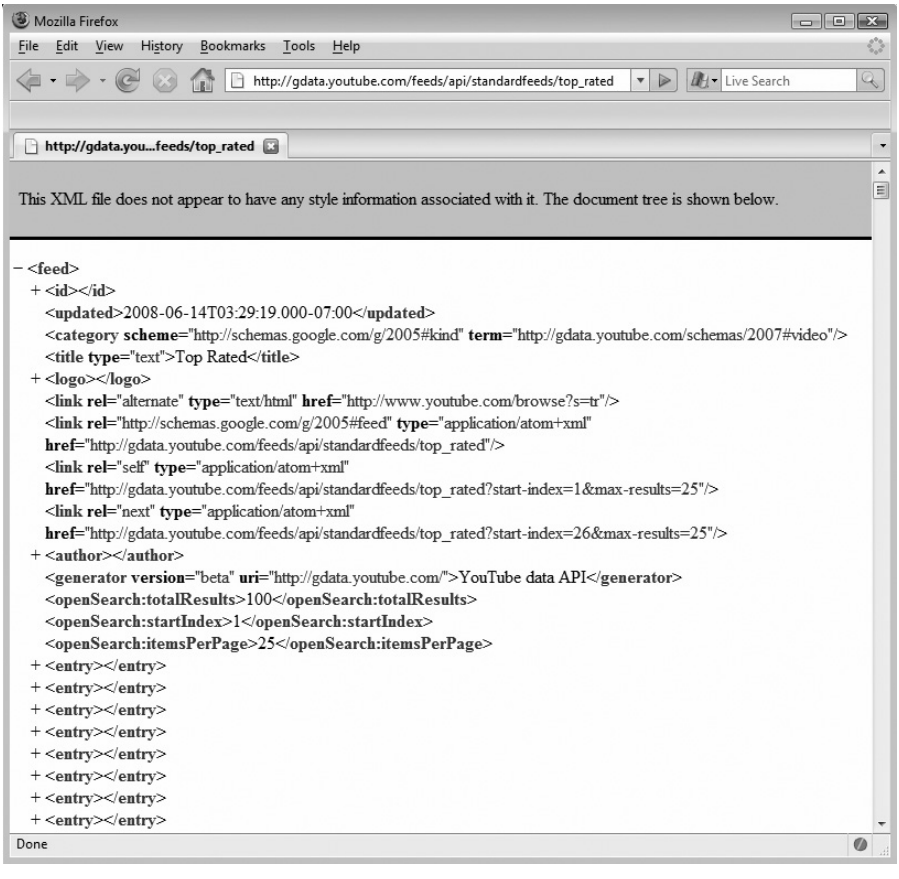

Рис. 4.4. *XML-документ, возвращаемый YouTube API*

Примечание

Полная документация о службах, предоставляемых YouTube, находится на странице *http://code.google.com/apis/youtube/developers_guide_protocol.html*.

Стандартные ленты YouTube

Приглядевшись к URL на рис. 4.4, вы заметите, что лента Top Rated относится к категории стандартных – «Standard Feed». На сайте YouTube определено несколько стандартных лент, в том числе:

- Top Rated (с наивысшим рейтингом): *http://gdata.youtube.com/feeds/api/standardfeeds/top_rated*
- Top Favorited (чаще всего помещаемые к папку «Избранное»): *http://gdata.youtube.com/feeds/api/standardfeeds/top_favorites*

- Most Viewed (чаще всего просматриваемые): *http://gdata.youtube.com/feeds/api/standardfeeds/most_viewed*

Вы можете ограничить множество роликов, извлекаемых из стандартной ленты, добавив в строку запроса параметр `time`. Например, в ответ на следующий запрос будут возвращены только ролики, которые чаще всего просматривались на этой неделе:

```
http://gdata.youtube.com/feeds/api/standardfeeds/most_viewed?time=this_week
```

Параметр `time` может принимать следующие значения: `today`, `this_week`, `this_month`, `all_time`. Если он не задан, то по умолчанию подразумевается `all_time` (за все время).

Связанные с пользователями ленты YouTube

По умолчанию YouTube раскрывает данные о пользователе (например, ваши избранные ролики и подписки). Они доступны с помощью API. Чтобы получить избранное или подписки некоторого пользователя, достаточно вместо строки *username* подставить в приведенный ниже URL имя существующего пользователя YouTube. Но в отличие от стандартных лент, вы не можете включить в такой URL параметр `time` и тем самым профильтровать связанную с пользователем ленту по времени.

- Избранное указанного пользователя: *http://gdata.youtube.com/feeds/api/users/username/favorites*
- Подписки указанного пользователя *http://gdata.youtube.com/feeds/api/users/username/ subscriptions*

Поиск по сайту YouTube

YouTube предоставляет также API поиска. Для обращения к нему следует включить в URL параметр vq=*searchterm*. Например, следующий URL описывает запрос на поиск всех роликов Microsoft:

```
http://gdata.youtube.com/feeds/api/videos?vq=Microsoft
```

Дополнительные параметры в строке запроса

Обертка, которую InnerTube реализует вокруг YouTube, поддерживает только простой поиск, но интерфейс самого YouTube включает также ряд сервисных механизмов, в частности разбиение на страницы и сортировку. Чтобы ими воспользоваться, необходимо включить в строку запроса дополнительные параметры. Ниже перечислены некоторые из них:

`orderby`

Задает порядок сортировки результатов, например: по релевантности, по рейтингу, по дате публикации или по количеству просмотров. По умолчанию подразумевается сортировка по релевантности.

`max-results`

Задает максимальное количество возвращаемых результатов. По умолчанию равно 25.

`start-index`

Режим разбиения на страницы. Параметр `start-index` должен быть равен порядковому номеру результата, с которого должна начинаться выборка. Если `start-index` равно 100, то API вернет результаты с 100 по 125. По умолчанию равен 0.

Примечание

Полный перечень параметров в строке запроса см. на странице *http://code.google.com/apis/youtube/reference.html#Query_parameter_definitions*.

Еще некоторые важные URL на сайте YouTube

На сайте YouTube есть ряд других проверенных URL, которые следует знать при создании приложения. Приведем их краткие описания:

Просмотр

Это ссылка, которая ведет непосредственно на видеоролик: *http://www.youtube.com/watch?v=VideoID*

Встраивание

Эта ссылка предназначена для размещения ролика на другом сайте, например MySpace: *http://www.youtube.com/v/VideoID*

Миниатюра

Это ссылка на миниатюру видео максимально возможного размера (425×344): *http://www.youtube.com/v/VideoID*

Скачивание

Это скрытая ссылка, которой мы будем пользоваться для скачивания ролика: *http://www.youtube.com/get_video?video_id=VideoID&t=sessiontoken*

Примечание

Ссылка для скачивания ролика будет работать, только если включить в URL корректный маркер сеанса, зависящий от времени. Это сделано для того, чтобы не дать пользователям скачивать ролики напрямую. Мы покажем, как обойти это ограничение, в разделе «Скачивание роликов с YouTube» ниже.

Классы для работы с роликами и лентами YouTube

Теперь, когда вы познакомились с устройством YouTube API, покажем, как представить типы возвращаемых API данных в виде классов .NET (рис. 4.5).

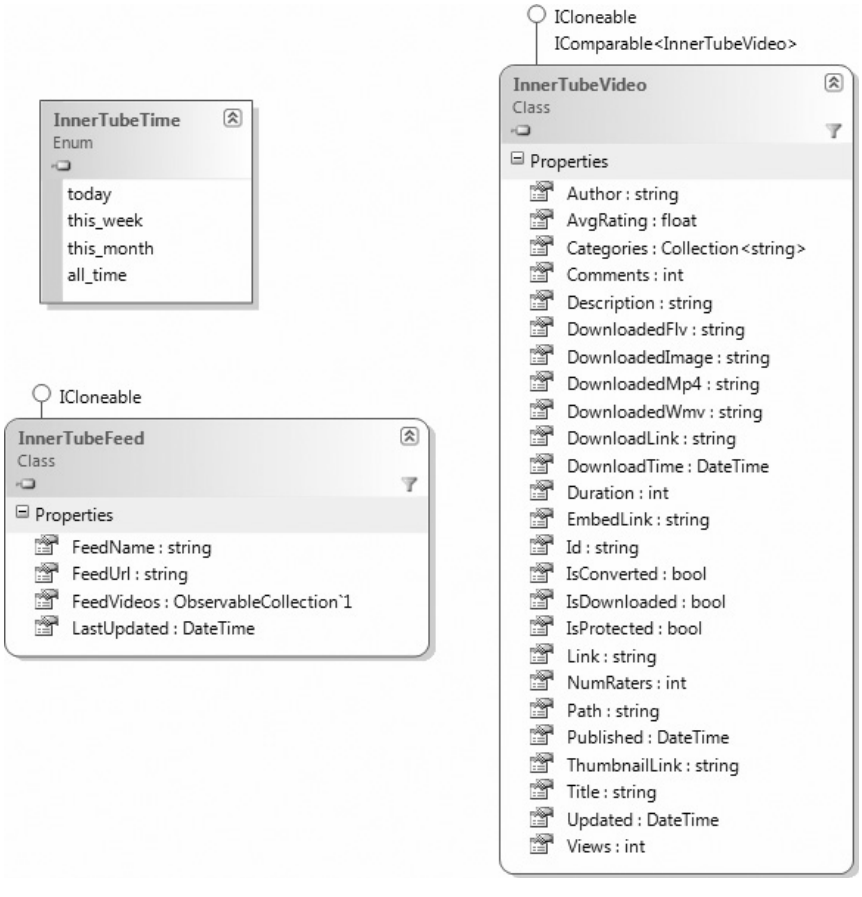

***Рис. 4.5.** Отображение YouTube API на классы .NET*

`InnerTubeTime`

Это перечисление представляет все возможные значения параметра `time`. Мы воспользуемся им в классе `InnerTubeService` для описания необязательного параметра, передаваемого YouTube API.

`InnerTubeFeed`

Этот класс представляет одну конкретную ленту, например Top Rated. Содержит понятное пользователю имя ленты `Feed Name` и URL, указывающий на один из перечисленных выше компонентов YouTube API.

`InnerTubeVideo`

Представляет один ролик YouTube и включает большую часть полей, возвращаемых YouTube (автор, комментарии, категории), а также свойства, относящиеся к уже скачанному ролику, например его местоположение на диске.

Строки или класс Uri?

Возможно, вы обратили внимание на то, что в классе `InnerTubeVideo` URL хранятся в виде строк, а не объектов класса Uri. Вообще говоря, для этой цели лучше использовать класс Uri, но InnerTube должна сериализовывать (уметь сохранять) свои классы на диск с помощью класса XmlSerializer. К сожалению, класс Uri не является сериализуемым и возбуждает исключение при попытке сериализации. Связано это с тем, что в классе Uri не определен конструктор без параметров.

Вызов класса InnerTubeService

Спроектировав классы, покажем, как с их помощью можно преобразовать XML-документ, изображенный на рис. 4.4, в набор объектов, представленных на рис. 4.5. Для этого воспользуемся еще одним классом: `InnerTubeService`. На рис. 4.6 видно, что стандартным лентам – Top

Рис. 4.6. *IntelliSense показывает методы для получения данных с YouTube*

Rated, Most Viewed и т. д. – соответствуют методы этого класса. Все они возвращают объект типа `ObservableCollection<InnerTubeVideo>`.

Зачем нужен класс ObservableCollection?

Сразу стоит отметить, что наборы видеороликов представляются классом `ObservableCollection`. Этот класс очень похож на `Collection`, но дополнительно реализует два события: `CollectionChanged` и `PropertyChanged`, которые упрощают обновление связанных элементов данных в WPF. Вопрос о связывании мы подробно обсудим в разделе «Анатомия MainWindow.xaml» ниже, а пока скажем лишь, что этот тип дает возможность автоматически обновлять элементы интерфейса при изменении набора, не требуя повторной привязки к данным.

Преобразование YouTube XML-документов в объекты .NET

По сути дела, в классе `InnerTubeService` есть всего один метод, который занимается преобразованием XML-документа в набор объектов, – `ConvertYouTubeXmlIntoObjects`. Объясняется это тем, что все возвращаемые YouTube API документы имеют одинаковую структуру, будь то результаты поиска или список роликов с наивысшим рейтингом. Поэтому и код для разбора любого XML-документа будет один и тот же, хотя мы все же включили ряд вспомогательных методов, например `Search(string query)`, чтобы упростить работу с библиотекой.

Из примеров 4.1 и 4.2 видно, что метод `ConvertYouTubeXmlToObjects` принимает два параметра: `Uri` некоторого компонента YouTube API и `Setting` – объект, описывающий конфигурацию каталогов на вашем ПК, в которых будут сохраняться различные файлы (видео, миниатюры и т. д.). Класс `Setting` мы подробнее рассмотрим ниже в разделе «Настройки приложения».

Пример 4.1. Сигнатура метода ConvertYouTubeXmlToObjects (версия на C#)

```
public static ObservableCollection<InnerTubeVideo>
ConvertYouTubeXmlToObjects(
  Uri youTubeUrl, Setting setting)
```

Пример 4.2. Сигнатура метода ConvertYouTubeXmlToObjects (версия на VB)

```
Public Shared Function ConvertYouTubeXmlToObjects(ByVal youTubeUrl As Uri, _
  ByVal setting As Setting) As ObservableCollection(Of InnerTubeVideo)
```

В теле метода `ConvertYouTubeXmlToObjects` мы объявляем несколько переменных типа `XNamespace`. Это необходимо, поскольку в получаемом от

YouTube документе широко используются пространства имен XML (см. примеры 4.3 и 4.4).

Пример 4.3. Объявление пространств имен XML (версия на C#)

```
XNamespace nsBase = @"http://www.w3.org/2005/Atom";
XNamespace nsGData = @"http://schemas.google.com/g/2005";
XNamespace nsYouTube = @"http://gdata.youtube.com/schemas/2007";
```

Пример 4.4. Объявление пространств имен XML (версия на VB)

```
Dim nsBase As XNamespace = "http://www.w3.org/2005/Atom"
Dim nsGData As XNamespace = "http://schemas.google.com/g/2005"
Dim nsYouTube As XNamespace = "http://gdata.youtube.com/schemas/2007"
```

Чтобы получить сами данные по сети, мы вызываем службу YouTube, пользуясь классом `WebClient`, который находится в пространстве имен `System.Net`. Мы обращаемся к методу `OpenRead()` этого класса, передавая ему URL службы. Результат будет точно такой же, как при получении XML-документа в броузере (см. рис. 4.5). Затем с помощью объекта `XmlTextReader` мы загружаем документ в объект типа `XDocument` (примеры 4.5 и 4.6).

Пример 4.5. Отправка веб-запроса YouTube API (версия на C#)

```
// Необходим для обращения к службе
WebClient wc = new WebClient();

// Получаем данные
XmlTextReader xr = new XmlTextReader(wc.OpenRead(youTubeUrl));
XDocument rawData = XDocument.Load(xr);
```

Пример 4.6. Отправка веб-запроса YouTube API (версия на VB)

```
' Необходим для обращения к службе
Dim wc As New WebClient()

' Получаем данные
Dim xr As New XmlTextReader(wc.OpenRead(youTubeUrl))
Dim rawData As XDocument = XDocument.Load(xr)
```

В этот момент мы имеем в памяти XML-документ, содержащий список видеороликов. Каждый элемент `<entry>` (один ролик) представлен в нем объектом типа `XElement`. Теперь мы хотим использовать технологию *LINQ for XML* для перебора всех элементов `<entry>` и создания объекта `InnerTubeVideo`, в котором будут установлены свойства, описывающие автора, категории, название и т. д. (см. примеры 4.7 и 4.8).

В первой строке в обоих примерах обратите внимание на то, что в качестве параметра методу `Descendents` передается `nsBase + "entry"`. Это необходимо, так как элемент `<entry>` находится в пространстве имен `nsBase` и без указания этого пространства имен LINQ-запрос возбудил бы исключение с сообщением о том, что такого атрибута у элемента не существует. Отметим также, что мы не строки конкатенируем, а используем

перегруженный оператор + в классе XNamespace. Именно поэтому в версии на Visual Basic используется оператор +, а не оператор конкатенации строк (&).

Пример 4.7. Цикл перебора элементов <entry> и установки свойств (версия на C#)

```
var query = from entry in rawData.Descendants(nsBase + "entry")
  select new InnerTubeVideo
  {
    Author = entry.Element(nsBase + "author").Element(nsBase + "name").Value,
    Categories = ParseCategories(entry.Elements(nsBase + "category")),
    Id = ParseID(entry.Element(nsBase + "id").Value),
    Published = DateTime.Parse(entry.Element(nsBase + "published").Value),
    Updated = DateTime.Parse(entry.Element(nsBase + "updated").Value),
    Title = entry.Element(nsBase + "title").Value,
    Description = entry.Element(nsBase + "content").Value,
...
```

Пример 4.8. Цикл перебора элементов <entry> и установки свойств (версия на VB)

```
Dim query = From entry In rawData.Descendants(nsBase + "entry") _
  Select New InnerTubeVideo With _
  {
    .Author = entry.Element(nsBase + "author").Element(nsBase +
              "name").Value, _
    .Categories = ParseCategories(entry.Elements(nsBase + "category")), _
    .Id = ParseID(entry.Element(nsBase + "id").Value), _
    .Published = DateTime.Parse(entry.Element(nsBase + "published").Value), _
    .Updated = DateTime.Parse(entry.Element(nsBase + "updated").Value), _
    .Title = entry.Element(nsBase + "title").Value, _
    .Description = entry.Element(nsBase + "content").Value, _...
```

Выше уже отмечалось, что некоторые интересующие нас поля находятся в различных пространствах имен, поэтому в программе нам приходится указывать подходящее пространство имен. Чтобы было понятнее, о чем идет речь, в примере 4.9 приведен фрагмент исходного XML-документа, полученного от YouTube API (содержит общее количество просмотров ролика).

Пример 4.9. Фрагмент XML-документа, содержащий общее количество просмотров ролика

```
<yt:statistics viewCount="130534" favoriteCount="1210" />
```

Поскольку атрибут viewCount находится в пространстве имен <yt> (сокращение от YouTube), то для получения его значения мы должны явно добавить переменную nsYouTube, в которой хранится ссылка на это пространство имен. В примерах 4.10 и 4.11 мы начинаем с корневого элемента entry, получаем вложенный в него элемент statistics и затем читаем атрибут viewCount.

Пример 4.10. Получение атрибута viewCount в XML-документе (версия на C#)

```
Views = int.Parse(entry.Element(nsYouTube +
          "statistics").Attribute("viewCount").Value),
```

Пример 4.11. Получение атрибута viewCount в XML-документе (версия на VB)

```
.Views = Integer.Parse(entry.Element(nsYouTube + _
          "statistics").Attribute("viewCount").Value),
```

У *LINQ for XML* есть удобная возможность, позволяющая сделать разбор XML-документа относительно компактным. Речь идет о передаче функции порции XML. Например, сногсшибательный ролик Джонни Ли (*http://www.youtube.com/watch?v=cI1AwZN4ZYg*) входит в 15 разных категорий YouTube, как показано в примере 4.12.

Пример 4.12. Фрагмент XML-документа, содержащий категории видеороликов

```
<category scheme="…" term="wiimote" />
<category scheme="…" term="display" />
<category scheme="…" term="3D" />
<category scheme="…" term="reality" />
<category scheme="…" term="nintendo" />
...
```

Мы хотим преобразовать атрибуты `term` элементов `<category>` в удобную для работы переменную типа `Collection<string>`, в которой каждый `term` будет представлен строкой `string`. Для этого мы вызываем метод `ParseCategories`, как показано в примерах 4.13 и 4.14.

Пример 4.13. Передача фрагмента XML методу ParseCategories (версия на C#)

```
Categories = ParseCategories(entry.Elements(nsBase + "category")),
```

Пример 4.14. Передача фрагмента XML методу ParseCategories (версия на VB)

```
.Categories = ParseCategories(entry.Elements(nsBase + "category")),
```

Метод `ParseCategories` получает на входе объект типа `IEnumerable<Xelement>`, который отформатирован точно так же, как элементы `<category>` в примере 4.12. Теперь мы можем воспользоваться LINQ, чтобы найти в каждом элементе `category` атрибут `term` и вернуть их в виде объекта `Collection<string>`, как показано в примерах 4.15 и 4.16.

Пример 4.15. Извлечение атрибутов term (версия на C#)

```
private static Collection<string> ParseCategories(IEnumerable<XElement>
        Categories)
{
  var vals = from c in Categories.Attributes("term")
```

```
    select c.Value;
  return (Collection<string>)vals;
}
```

Пример 4.16. Извлечение атрибутов term (версия на VB)

```
Private Shared Function ParseCategories(ByVal Categories As
        IEnumerable(Of XElement))
As Collection(Of String)
  Dim vals = From c In Categories.Attributes("term") _
  Select c.Value
  Return vals.ToCollection()
End Function
```

В примерах 4.17 и 4.18 приведен полный текст метода `ConvertYouTube-XmlToObjects`.

Пример 4.17. Метод ConvertYouTubeXmlToObjects (версия на C#)

```
public static ObservableCollection<InnerTubeVideo>
ConvertYouTubeXmlToObjects(
  Uri youTubeUrl, Setting setting)
{
  XNamespace nsBase = @"http://www.w3.org/2005/Atom";
  XNamespace nsGData = @"http://schemas.google.com/g/2005";
  XNamespace nsYouTube = @"http://gdata.youtube.com/schemas/2007";

  // Использовать для обращения к службе
  WebClient wc = new WebClient();

  // Получить данные
  XmlTextReader xr = new XmlTextReader(wc.OpenRead(youTubeUrl));
  XDocument rawData = XDocument.Load(xr);
  var query = from entry in rawData.Descendants(nsBase + "entry")
  select new InnerTubeVideo
  {
    Author = entry.Element(nsBase + "author").Element(nsBase + "name").Value,
    Categories = ParseCategories(entry.Elements(nsBase + "category")),
    Id = ParseID(entry.Element(nsBase + "id").Value),
    Published = DateTime.Parse(entry.Element(nsBase + "published").Value),
    Updated = DateTime.Parse(entry.Element(nsBase + "updated").Value),
    Title = entry.Element(nsBase + "title").Value,
    Description = entry.Element(nsBase + "content").Value,
    ThumbnailLink = _BaseThumbnailUrl + ParseID(entry.Element(nsBase +
                    "id").Value) + @"/0.jpg",
    Link = _BasewatchUrl + ParseID(entry.Element(nsBase + "id").Value),
    EmbedLink = _baseEmbedUrl + ParseID(entry.Element(nsBase + "id").Value),
    DownloadLink = _BaseDownloadUrl + ParseID(entry.Element(nsBase +
       "id").Value),
    Views = int.Parse(entry.Element(nsYouTube +
       "statistics").Attribute("viewCount").Value),
    AvgRating = float.Parse(entry.Element(nsGData +
       "rating").Attribute("average").Value),
    NumRaters = int.Parse(entry.Element(nsGData +
```

```
        "rating").Attribute("numRaters").Value),

    // установить, где сохранять скачанные файлы
    DownloadedImage = FileHelper.BuildFileName(setting.SubPath,
      ParseID(entry.Element(nsBase + "id").Value), FileType.Image),
    DownloadedFlv = FileHelper.BuildFileName(setting.SubPath,
      entry.Element(nsBase + "title").Value, FileType.Flv),
    DownloadedMp4 = FileHelper.BuildFileName(setting.VideoPath,
      entry.Element(nsBase + "title").Value, FileType.Mp4),
    DownloadedWmv = FileHelper.BuildFileName(setting.VideoPath,
      entry.Element(nsBase + "title").Value, FileType.Wmv)
  };

  return query.ToObservableCollection<InnerTubeVideo>();
}
```

Пример 4.18. Метод ConvertYouTubeXmlToObjects (версия на VB)

```
Public Shared Function ConvertYouTubeXmlToObjects(ByVal youTubeUrl As Uri, _
  ByVal setting As Setting) As ObservableCollection(Of InnerTubeVideo)

  Dim nsBase As XNamespace = "http://www.w3.org/2005/Atom"
  Dim nsGData As XNamespace = "http://schemas.google.com/g/2005"
  Dim nsYouTube As XNamespace = "http://gdata.youtube.com/schemas/2007"

  ' Использовать для обращения к службе
  Dim wc As New WebClient()

  ' Получить данные
  Dim xr As New XmlTextReader(wc.OpenRead(youTubeUrl))
  Dim rawData As XDocument = XDocument.Load(xr)
  Dim query = From entry In rawData.Descendants(nsBase + "entry") _

  Select New InnerTubeVideo With _
  { _
    .Author = entry.Element(nsBase + "author").Element(nsBase +
        "name").Value, _
    .Categories = ParseCategories(entry.Elements(nsBase + "category")), _
    .Id = ParseID(entry.Element(nsBase + "id").Value), _
    .Published = DateTime.Parse(entry.Element(nsBase + "published").Value), _
    .Updated = DateTime.Parse(entry.Element(nsBase + "updated").Value), _
    .Title = entry.Element(nsBase + "title").Value, _
    .Description = entry.Element(nsBase + "content").Value, _
    .ThumbnailLink = _BaseThumbnailUrl & ParseID(entry.Element(nsBase + _
      "id").Value)& "/0.jpg", _
    .Link = _BasewatchUrl & ParseID(entry.Element(nsBase + "id").Value), _
    .EmbedLink = _baseEmbedUrl & ParseID(entry.Element(nsBase + "id").Value), _
    .DownloadLink = _BaseDownloadUrl & ParseID(entry.Element(nsBase +
        "id").Value), _
    .Views = Integer.Parse(entry.Element(nsYouTube + _
        "statistics").Attribute("viewCount").Value), _
    .AvgRating = Single.Parse(entry.Element(nsGData + _
        "rating").Attribute("average").Value), _
    .NumRaters = Integer.Parse(entry.Element(nsGData + _
      "rating").Attribute("numRaters").Value), _
```

```
            .DownloadedImage = FileHelper.BuildFileName(setting.SubPath, _
              ParseID(entry.Element(nsBase + "id").Value), FileType.Image), _
            .DownloadedFlv = FileHelper.BuildFileName(setting.SubPath,
              entry.Element(nsBase + "title").Value, FileType.Flv), _
            .DownloadedMp4 = FileHelper.BuildFileName(setting.VideoPath, _
              entry.Element(nsBase + "title").Value, FileType.Mp4), _
            .DownloadedWmv = FileHelper.BuildFileName(setting.VideoPath, _
              entry.Element(nsBase + "title").Value, FileType.Wmv) _
        }

        Return query.ToObservableCollection()

    End Function
```

Теперь, когда мы научились получать информацию о размещенных на YouTube роликах, посмотрим, как можно скачать ролик из программы.

Скачивание видеороликов с YouTube

Сейчас мы покажем, как скачивать с YouTube видеоролики – сначала вручную с помощью броузера, а потом и программно.

Скачивание ролика с помощью броузера

На сайте YouTube видеролики хранятся в формате Flash video (FLV). Именно в таком формате мы и будем их получать.

Но прежде чем программировать скачивание, посмотрим, как это можно сделать с помощью броузера. Для начала откройте любую страницу YouTube с роликом, например знаменитый ролик Тэя Зондея «Chocolate Rain» (Шоколадный дождь) (*http://www.youtube.com/watch?v=EwTZ2xpQwpA*). Из этого URL следует, что идентификатор ролика (параметр v) равен «EwTZ2xpQwpA».

YouTube вас не пускает (в соответствии с проектным решением)

Существует (скрытый) URL, позволяющий скачать видео напрямую, но в нем нужно указать два параметра: `video_id` и *маркер сеанса*. Маркер сеанса – это идентификатор, который YouTube назначает вашему броузеру; он действителен в течение примерно 15 минут. Если у вас нет действующего маркера сеанса, то YouTube отклонит запрос на скачивание Flash-ролика.

Чтобы убедиться в этом, откройте свой броузер и перейдите на страницу *http://www.youtube.com/get_video?video_id=EwTZ2xpQwpA*. Смотрите-ка – YouTube притворяется, будто получил негодный URL, он возвращает броузеру Internet Explorer ответ `HTTP 404 Not Found` (Страница не найдена) (пользователи Firefox 3 увидят пустую страницу) (рис. 4.7). Все дело в том, что в URL следует включить действительный маркер сеанса.

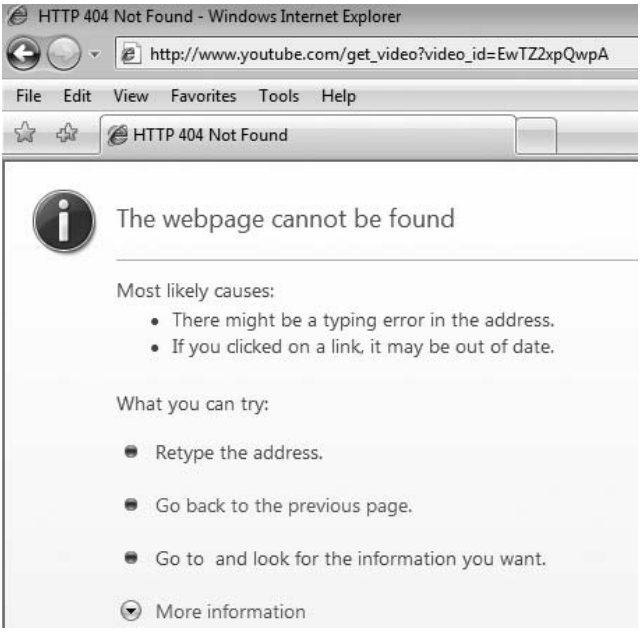

Рис. 4.7. *YouTube возвращает броузеру Internet Explorer ошибку 404, поскольку вы не указали маркер сеанса*

Получение маркера сеанса с помощью JavaScript

Чтобы получить от YouTube действительный маркер сеанса, откройте в броузере страницу с роликом «Chocolate Rain» по адресу *http://www.youtube.com/watch?v=EwTZ2xpQwpA* и выберите из меню команду «Просмотр HTML-кода». Найдите в HTML-коде переменную JavaScript с именем `swfArgs`, в которой YouTube передает параметры Flash-плееру (пример 4.19).

Пример 4.19. Javascript-переменная swfArgs используется для настройки видеоплеера

```
var swfArgs = {"usef": 0, "cust_p": "jMWn75PwKgutJQOJ3mrLbA", "iv_storage_
server": "http://www.google.com/reviews/y/", "ad_module":
         "http://s.ytimg.com/yt/
swf/ad-vfl59966.swf", "ad_channel_code": "
invideo_overlay_480x70_cat10,afv_overlay",
"video_id": "EwTZ2xpQwpA", "l": 292, "fmt_map": "34/0/9/0/115", "ad_host":
"ca-host-pub-5311789755034317", "sk": "_N53QD2GOB79IwT2MIi7nNSvpkgWSWWwC",
"invideo": true, "t": "OEgsToPDskJUt8xv3hrKiGOAYlLYcl1L", "hl": "en", "plid":
"AARaIAIC2yUqgjl1AAAA-YT8YQA", "vq": null, "iv_module":
         "http://s.ytimg.com/yt/swf/
```

```
iv_module-vfl57703.swf", "afv": true, "ad_host_tier": "12789",
        "ad_video_pub_id":
"ca-pub-6219811747049371", "tk": "5Eu9v6C5n2lxivxOt0UqRqT0yNnUmUCs3oR7ZzuDg8_
JZHd6DyO2jw=="};
```

Ага, вот и маркер!

Глубоко внутри переменной `swfArgs` закопана пара имя/значение `"t": "OEgsToPDskJUt8xv3hrKiGOAYlLYcl1L"`, которая и дает действительный маркер сеанса, нужный для скачивания ролика. Давайте попробуем взять *тот же самый* URL, для которого раньше получали ошибку 404, но на этот раз добавим в него параметр `t`, как показано ниже:

```
http://www.youtube.com/get_video?video_id=EwTZ2xpQwpA&t=OEgsToPDskJUt8
xv3hrKiGOAYlLYcl1L
```

Примечание

Именно этот URL не подойдет, поскольку к тому моменту, когда вы будете читать этот текст, срок действия маркера (параметр t=) истечет. Если вы хотите провести эксперимент со своим броузером, то сначала зайдите на какую-нибудь страницу YouTube с роликом, откройте ее исходный текст и вручную скопируйте маркер сеанса в URL скачивания. Если вы попробуете воспользоваться тем же маркером по истечении срока его действия, то получите ошибку HTTP 410 Gone, означающую, что такая страница уже не существует. Маркеры сеанса привязаны также и к роликам, то есть одним и тем же маркером не удастся воспользоваться для скачивания разных роликов.

Теперь появится диалоговое окно с предложением сохранить FLV-файл размером 11.3 Мбайт на диске (рис. 4.8). Посмотрим, как то же самое сделать из программы.

Получение маркера из программы

Поняв, как добиться желаемого с помощью броузера, мы хотели бы имитировать всю процедуру получения маркера программно. Для этого следует прочитать HTML-страницу, содержащую ролик, найти в ней JavaScript-переменную `swfArgs`, а затем обратиться по адресу скачивания, подставив в него действительный маркер.

Примечание

Описанный ниже алгоритм разбора HTML-кода неустойчив. Он перестанет работать, стоит YouTube изменить механизм работы с маркерами сеансов или назвать Javascript-переменные по-другому. Принимая во внимание, что API для скачивания роликов не предоставляется, разработчики приложений, зависящих от недокументированной структуры HTML-разметки сайта, должны постоянно тестировать свои программы и стремиться к тому, чтобы изменение сайта не стало причиной неработоспособности программ.

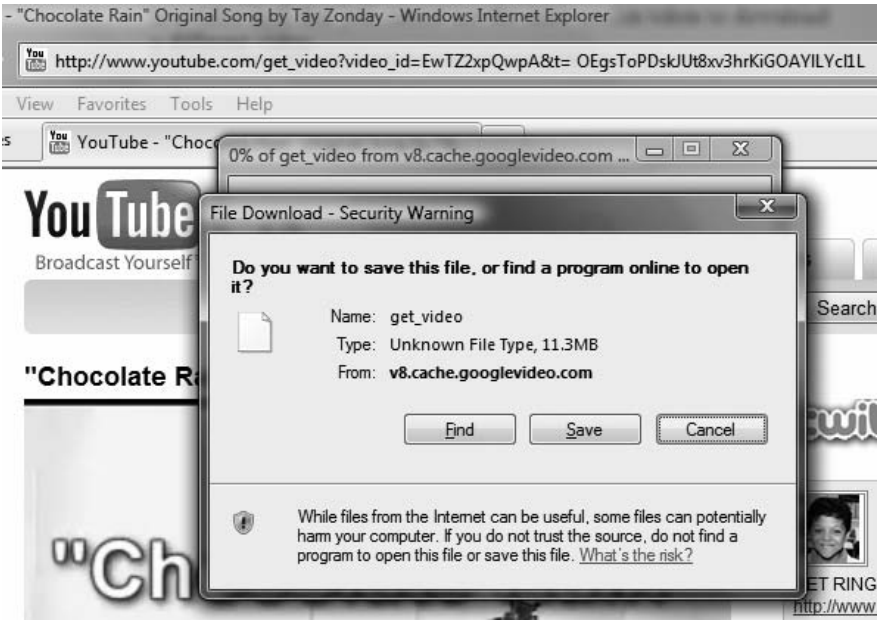

Рис. 4.8. *Предложение сохранить ролик с YouTube на локальном диске*

Чтобы сделать это из программы, мы включили в класс `Download` метод `CreateTokenRequest`. Если передать ему объект типа `InnerTubeVideo`, то он отправит с помощью `WebClient` запрос на скачивание HTML-кода страницы с роликом. HTML-код будет получен в виде строки. Из этой строки метод выделит JavaScript-переменную `swfArgs`, а в ней найдет аргумент `"t" :`, как мы вручную поступали в броузере.

Код в примерах 4.20 и 4.21 получает HTML-код веб-страницы, обращаясь к методу `DownloadString` класса `WebClient`, которому передается свойство `Link` (URL видеоролика на сайте YouTube). Затем мы получаем позицию, с которой начинается переменная `swfArgs` (см. пример 4.19) в полученной строке. Зная смещение `swfArgs`, мы можем получить значение этой переменной; для этого требуется найти фигурные скобки «{»и «}», внутри которых заключено значение. Получив индексы обеих скобок, мы с помощью метода `SubString` извлекаем строку, находящуюся между ними, и присваиваем ее переменной `fullString`.

Пример 4.20. Программное получение маркера сеанса (версия на C#)

```
private static string CreateTokenRequest(InnerTubeVideo video)
{
  // переменные YouTube
  const string jsVariable = "swfArgs";
  const string argName = "t";
```

```
  // получить HTML-код страницы YouTube с роликом
  string rawHtml;
  using (WebClient wc = new WebClient())
  {
    rawHtml = wc.DownloadString(video.Link);
  }

  // выделить из Javascript пары имя/значение
  int jsIndex = rawHtml.IndexOf(jsVariable);
  int startIndex = rawHtml.IndexOf("{", jsIndex);
  int endIndex = rawHtml.IndexOf("}", startIndex);
  string fullString = rawHtml.Substring(startIndex + 1, endIndex
    - startIndex - 1);

  // удалить все кавычки (")
  fullString = fullString.Replace("\"", "");

  // выделить все значения
  string[] allArgs = fullString.Split(',');

  // обойти все параметры javascript
  foreach (string swfArg in allArgs)
  {
    if (swfArg.Trim().StartsWith(argName))
    {
      var nameValuePair = swfArg.Split(':');
      return string.Format("{0}={1}", argName, nameValuePair[1].Trim());
    }
  }
  throw new Exception(string.Format("token not found in swfArgs," +
    " make sure {0} is accessible", video.Link));
}
```

Пример 4.21. Программное получение маркера сеанса (версия на VB)

```
Private Shared Function CreateTokenRequest(ByVal video As InnerTubeVideo)
    As String
  ' переменные YouTube
  Const jsVariable As String = "swfArgs"
  Const argName As String = "t"

  ' получить HTML-код страницы YouTube с роликом
  Dim rawHtml As String
  Using wc As New WebClient()
    rawHtml = wc.DownloadString(video.Link)
  End Using

  ' выделить из Javascript пары имя/значение
  Dim jsIndex As Integer = rawHtml.IndexOf(jsVariable)
  Dim startIndex As Integer = rawHtml.IndexOf("{", jsIndex)
  Dim endIndex As Integer = rawHtml.IndexOf("}", startIndex)
  Dim fullString As String = rawHtml.Substring(startIndex + 1, _
    endIndex - startIndex - 1)

  ' удалить все кавычки (")
  fullString = fullString.Replace("""", "")
```

```
    ' выделить все значения
    Dim allArgs() As String = fullString.Split(",c)

    ' обойти все параметры javascript
    For Each swfArg As String In allArgs
      If swfArg.Trim().StartsWith(argName) Then
        Dim nameValuePair = swfArg.Split(":c)
        Return String.Format("{0}={1}", argName, nameValuePair(1).Trim())
      End If
    Next swfArg
    Throw New Exception(String.Format("token not found in swfArgs, " & _
      "make sure {0} is accessible", video.Link))
  End Function
```

Поместив в переменную `fullString` значение `swfArgs`, мы должны затем выделить маркер сеанса. Для этого удалим из строки все кавычки и разобьем значение переменной на части (аргументы отделяются друг от друга запятыми). Наконец, переберем в цикле все аргументы и найдем среди них аргумент `"t"` (маркер сеанса). Отыскав его, вернем строку в формате "имя=значение". Для данного ролика она будет выглядеть следующим образом:

```
t=OEgsToPDskJUt8xv3hrKiGOAYlLYcl1L
```

Скачивание видео

Решив проблему программного получения маркера сеанса, мы должны теперь скачать файл и записать его к себе на диск. Для этого воспользуемся методом `DownloadVideo` класса `Download`, полный текст которого приведен в примерах 4.22 и 4.23.

Первым делом убедимся, что интересующий нас Flash-ролик (FLV-файл) не был скачан ранее. Для этого проверим, есть ли уже на диске соответствующий файл. Затем сконструируем URL скачивания, который, как и при работе с броузером, состоит из URL ресурса, идентификатора ролика и маркера сеанса, только что полученного нами от метода `CreateTokenRequest`.

Имея URL, мы отправляем запрос серверу YouTube и открываем поток, из которого будем читать видеофайл. В то же время мы создаем объект `FileStream`, соответствующий дисковому файлу, где собираемся сохранить ролик. Наша программа читает данные из веб-потока блоками по 64 Кбайт и записывает их в файловый поток. Цикл продолжается, пока не будет достигнут конец файла, то есть метод `ReadBytes` не вернет 0.

Примечание

Объекты `WebStream` и `FileStream` помещены внутрь предложений `using`. Тем самым мы автоматически добиваемся корректного освобождения ресурсов в генерируемом компилятором блоке try/finally. Это необходимо для того, чтобы даже в случае исключения системные ресурсы освобождались вовремя.

Пример 4.22. Метод DownloadVideo (версия на C#)

```
public static void DownloadVideo(InnerTubeVideo source, string destination)
{
  if (!File.Exists(destination))
  {
    UriBuilder final = new UriBuilder(source.DownloadLink);
    final.Query = "video_id=" + source.Id + "&" + CreateTokenRequest(source);

    WebRequest request = WebRequest.Create(final.ToString());
    request.Timeout = 500000;

    try
    {
      WebResponse response = request.GetResponse();

      using (Stream webStream = response.GetResponseStream())
      {
        try
        {
          int _bufferSize = 65536;

          using (FileStream fs = File.Create(destination, _bufferSize))
          {
            int readBytes = -1;
            byte[] inBuffer = new byte[_bufferSize];

            // цикл до обнаружения конца файла
            while (readBytes != 0)
            {
              // читаем данные из сети в filebuffer, затем пишем в файл
              readBytes = webStream.Read(inBuffer, 0, _bufferSize);
              fs.Write(inBuffer, 0, readBytes);
            }
          }
        }
        catch (Exception ex)
        {
          Debug.WriteLine("Error in Buffer Download");
          Debug.Indent();
          Debug.WriteLine(ex.Message);
        }
      }
    }
    catch (Exception ex)
    {
      Debug.WriteLine("Error in request.GetResponse()");
      Debug.Indent();
      Debug.WriteLine(ex.Message);
    }
  }
}
```

Пример 4.23. Метод DownloadVideo (версия на VB)

```
Public Shared Sub DownloadVideo(ByVal source As InnerTubeVideo,
    ByVal destination As String)
  If (Not File.Exists(destination)) Then
    Dim final As New UriBuilder(source.DownloadLink)
    final.Query = "video_id=" & source.Id & "&" & CreateTokenRequest(source)

    Dim request As WebRequest = WebRequest.Create(final.ToString())
    request.Timeout = 500000

    Try
      Dim response As WebResponse = request.GetResponse()

      Using webStream As Stream = response.GetResponseStream()
        Try
          Dim _bufferSize As Integer = 65536

          Using fs As FileStream = File.Create(destination, _bufferSize)
            Dim readBytes As Integer = -1
            Dim inBuffer(_bufferSize - 1) As Byte

            ' цикл до обнаружения конца файла
            Do While readBytes <> 0
              ' читаем данные из сети в filebuffer, затем пишем в файл
              readBytes = webStream.Read(inBuffer, 0, _bufferSize)
              fs.Write(inBuffer, 0, readBytes)
            Loop
          End Using
          Catch ex As Exception
            Debug.WriteLine("Error in Buffer Download")
            Debug.Indent()
            Debug.WriteLine(ex.Message)
        End Try
      End Using
    Catch ex As Exception
      Debug.WriteLine("Error in request.GetResponse()")
      Debug.Indent()
      Debug.WriteLine(ex.Message)
    End Try
  End If
End Sub
```

Скачивание заставки видеоролика

Помимо скачивания самого ролика, InnerTube скачивает еще и большую, размером 425×344 пиксела, миниатюру, которая будет отображаться в качестве заставки перед началом воспроизведения видео. URL большой миниатюры имеет вид *http://img.youtube.com/vi/VideoID/0.jpg*.

Код, скачивающий изображение (примеры 4.24 и 4.25), просто проверяет, был ли файл скачан ранее, и, если это не так, то вызывает метод `DownloadFile`, передавая его адрес в сети и имя конечного файла.

Пример 4.24. Скачивание миниатюрной заставки ролика (версия на C#)

```
public static void DownloadImage(InnerTubeVideo source, string destination)
{
  // если изображение еще не скачивалось, сделаем это сейчас
  if (!File.Exists(destination))
  {
    using (WebClient wc = new WebClient())
    {
      wc.DownloadFile(new Uri(source.ThumbnailLink), destination);
    }
  }
}
```

Пример 4.25. Скачивание миниатюрной заставки ролика (версия на VB)

```
Public Shared Sub DownloadImage(ByVal source As InnerTubeVideo,
       ByVal destination As String)
  ' если изображение еще не скачивалось, сделаем это сейчас
  If (Not File.Exists(destination)) Then
    Using wc As New WebClient()
      wc.DownloadFile(New Uri(source.ThumbnailLink), destination)
    End Using
  End If
End Sub
```

Классы WebRequest и WebClient

Как вы, наверное, заметили, для скачивания ролика мы пользовались классом `WebRequest`, а для скачивания заставки – классом `WebClient`. Сравнение кода наглядно показывает, что работать с классом `WebClient` проще, поскольку не требуется производить сложные манипуляции с потоком и массивом байтов, как в случае `WebRequest`. Собственно говоря, в первых версиях InnerTube в методе `DownloadVideo` тоже использовался класс `WebClient`, но оказалось, что при скачивании видео он время от времени возбуждает исключение `WebException`, сообщая, что произошел тайм-аут. Поскольку `WebClient` не позволяет задать величину тайм-аута запроса, пришлось прибегнуть к классу `WebRequest` и задавать большой тайм-аут (50K миллисекунд или 8.3 минут), чтобы избежать подобных ошибок.

Конвертирование роликов YouTube с помощью программы ffmpeg

Итак, ролик находится у нас на диске в формате Flash (FLV). Мы, конечно, могли бы создать простенькую HTML-страничку и включить в нее Flash-ролик для локального просмотра, но по-настоящему хотелось бы конвертировать видео в формат WMV (для Zune) или MP4 (для iTunes). Для этой цели мы воспользуемся инструментом `ffmpeg` с открытыми исходными текстами, который представляет собой совокупность написанной на C++ библиотеки и командной утилиты для конвертирования мультимедийных файлов из одного формата в другой с возможностью масштабирования. Поддерживается более 50 форматов.

В самой простой форме программа *ffmpeg.exe* вызывается с двумя аргументами: имя исходного файла, подлежащего конвертированию, и имя результирующего файла. Например, для конвертирования Flash-ролика *source.flv* в формат MP4 (MPEG-4) с записью в файл *destination.mp4* можно воспользоваться командой, приведенной в примере 4.26.

Пример 4.26. Конвертирование Flash-ролика в формат MP4

```
ffmpeg.exe -i "source.flv" "destination.mp4"
```

Для конвертирования файла *source.flv* в формат WMV (Windows Media Video) добавьте флаг `-vcodec wmv2` (пример 4.27).

Пример 4.27. Конвертирование Flash-ролика в формат WMV

```
ffmpeg.exe -i "source.flv" -vcodec wmv2 "destination.wmv"
```

Можно даже включить в конвертированный файл метаданные – имя автора, заголовок и комментарий. Для этого нужно задать в командной строке *ffmpeg.exe* дополнительные флаги, как показано в примере 4.28:

Пример 4.28. Задание метаданных при конвертировании видеофайла

```
ffmpeg.exe -title "My Video Title" -author "Dan" -comment "best video ever" -i
"source.flv" -vcodec wmv2 "destination.wmv"
```

Чтобы вызвать *ffmpeg.exe* из программы, мы сначала определим перечисление `ConversionType`, описывающее, какие виды конвертирования допускаются в нашем приложении (см. примеры 4.29 и 4.30).

Пример 4.29. Перечисление ConversionType (версия на C#)

```
public enum ConversionType
{
  Mp4,
  Wmv
}
```

Пример 4.30. Перечисление ConversionType (версия на VB)

```
Public Enum ConversionType
  Mp4
  Wmv
End Enum
```

Для вызова консольного приложения *ffmpeg.exe* из программы мы воспользуемся классом `System.Diagnostics.Process`. Вызов *ffmpeg* обернут в класс `VideoConverter`, в котором есть всего один перегруженный метод `ConvertFlv`. Тот вариант `ConvertFlv`, который нас интересует, принимает объект типа `InnerTubeVideo` и одно из значений перечисления `ConversionType` (MP4 или WMV). Мы воспользуемся именно этим вариантом, поскольку хотим вставить название, имя автора и описание ролика в виде метаданных, как показано в примере 4.28. Так как эти поля могут содержать недопустимые символы, из-за которых *ffmpeg.exe* откажется работать, мы предварительно вычистим их методом `ReplaceIllegalCharacters`.

Как видно из примеров 4.31 и 4.32, мы конструируем аргументы командной строки для передачи *ffmpeg.exe*, исходя из того, какую конвертацию хотим произвести (в MP4 или WMV).

Пример 4.31. Задание параметров ffmpeg (версия на C#)

```
public static void ConvertFlv(InnerTubeVideo source, ConversionType
conversion)
{
  string title = FileHelper.ReplaceIllegalCharacters(source.Title);
  string author = FileHelper.ReplaceIllegalCharacters(source.Author);
  string description =
FileHelper.ReplaceIllegalCharacters(source.Description);

// задать значения, соответствующие режиму конвертирования
string cmdLineArgs = String.Empty;
string destination = String.Empty;

switch (conversion)
{
  case ConversionType.Mp4:
  //ffmpeg.exe -title "Chocolate Rain" -author "TayZonday" -comment "Original
  //Song by Tay Zonday" -i "Chocolate Rain.flv" "Chocolate Rain.mp4"
    destination = source.DownloadedMp4;
    cmdLineArgs = String.Format(" -title \"{0}\" -author \"{1}\"
       -comment \"{2}\" -i \"{3}\" \"{4}\"",title, author, description,
       source.DownloadedFlv, destination);
  break;
  case ConversionType.Wmv:
  //ffmpeg.exe -title "Chocolate Rain" -author "TayZonday" -comment "Original
  //Song by Tay Zonday" -i "Chocolate Rain.flv" -vcodec wmv2
  //"Chocolate Rain.wmv"
    destination = source.DownloadedWmv;
    cmdLineArgs = String.Format(" -title \"{0}\" -author \"{1}\"
```

```
        -comment \"{2}\" -i \"{3}\" -vcodec wmv2 \"{4}\"", title,
        author, description, source.DownloadedFlv, destination);
      break;
    }
    ConvertFlv(source.DownloadedFlv, destination, cmdLineArgs);
  }
```

Пример 4.32. Задание параметров ffmpeg (версия на VB)

```
Public Shared Sub ConvertFlv(ByVal source As InnerTubeVideo, _
        ByVal conversion As ConversionType)
  Dim title As String = FileHelper.ReplaceIllegalCharacters(source.Title)
  Dim author As String = FileHelper.ReplaceIllegalCharacters(source.Author)
  Dim description As String = _
        FileHelper.ReplaceIllegalCharacters(source.Description)

  ' задать значения, соответствующие режиму конвертирования
  Dim cmdLineArgs As String = String.Empty
  Dim destination As String = String.Empty
  Select Case conversion
    Case ConversionType.Mp4
      'ffmpeg.exe -title "Chocolate Rain" -author "TayZonday" _
          -comment "Original Song
      'by Tay Zonday" -i "Chocolate Rain.flv" "Chocolate Rain.mp4"
      destination = source.DownloadedMp4
      cmdLineArgs = String.Format(" -title ""{0}"" -author ""{1}"" " & _
        "-comment ""{2}"" -i ""{3}"" ""{4}""", title, author, _
        description, source.DownloadedFlv, destination)
    Case ConversionType.Wmv
     'ffmpeg.exe -title "Chocolate Rain" -author "TayZonday" _
          -comment "Original Song
     'by Tay Zonday" -i "Chocolate Rain.flv" -vcodec wmv2 "Chocolate _
          Rain.wmv"
      destination = source.DownloadedWmv
      cmdLineArgs = String.Format(" -title ""{0}"" -author ""{1}"" -comment " _
      & " ""{2}"" -i ""{3}"" -vcodec wmv2 ""{4}""", title, author, _
      description, source.DownloadedFlv, destination)
  End Select
  ConvertFlv(source.DownloadedFlv, destination, cmdLineArgs)
End Sub
```

Вы, конечно, заметили, что в последней строке в примерах 4.31 и 4.32 вызывается еще один перегруженный вариант метода `ConvertFlv`, который собственно и запускает *ffmpeg* из командной строки. Этот вариант мы сейчас и обсудим.

В примерах 4.33 и 4.34 в коде метода `ConvertFlv` фигурирует местоположение исполняемого файла *ffmpeg.exe*. При установке этот файл помещается в папку *ffmpeg* каталога *SharedUtilities*, поэтому мы можем воспользоваться свойством `Environment.CurrentDirectory` для получения полного пути к исполняемому файлу и сохранить этот путь в переменной `exePath`.

Мы также проверяем, что исходный файл существует, а конечный – нет. В предположении, что все готово для работы, мы создаем процесс, устанавливаем некоторые его свойства, в частности аргументы командной строки, местоположение EXE-файла и флаги, необходимые для того, чтобы консольное окно не появлялось на экране во время конвертирования. И наконец вызываем метод `Start` для запуска конвертера. Поскольку `ffmpeg.exe` автоматически завершается после выполнения конвертирования, мы обращаемся к методу `WaitForExit`, то есть метод `ConvertFlv` не вернет управление, пока конвертирование не будет завершено.

Пример 4.33. Создание процесса и вызов ffmpeg.exe (версия на C#)

```
private static void ConvertFlv(string sourceFile, string destination,
  string cmdLineArgs)
{
  // указывает на программу ffmpeg
  string exePath = Path.Combine(Environment.CurrentDirectory,
        @"ffmpeg\ffmpeg.exe");

  // проверим, что исходный файл существует, а конечный - нет
  if (File.Exists(sourceFile) && File.Exists(exePath) &&
        !File.Exists(destination))
  {
    // создаем и запускаем процесс для запуска внешней
    // утилиты конвертирования
    using (Process convert = new Process())
    {
      // устанавливаем свойства
      convert.StartInfo.WindowStyle = ProcessWindowStyle.Hidden;
      convert.StartInfo.CreateNoWindow = true;
      convert.StartInfo.RedirectStandardOutput = true;
      convert.StartInfo.UseShellExecute = false;
      convert.StartInfo.Arguments = cmdLineArgs;
      convert.StartInfo.FileName = exePath;
      convert.Start();
      convert.WaitForExit();
    }
  }
}
```

Пример 4.34. Создание процесса и вызов ffmpeg.exe (версия на VB)

```
Private Shared Sub ConvertFlv(ByVal sourceFile As String, ByVal
        destination As _ String, ByVal cmdLineArgs As String)
  ' указывает на программу ffmpeg
  Dim exePath As String = Path.Combine(Environment.CurrentDirectory, _
    "ffmpeg\ffmpeg.exe")

  ' проверим, что исходный файл существует, а конечный - нет
  If File.Exists(sourceFile) AndAlso File.Exists(exePath) AndAlso _
    (Not File.Exists(destination)) Then
```

```
        ' создаем и запускаем процесс для запуска внешней
        ' утилиты конвертирования
        Using convert As New Process()
          ' устанавливаем свойства
          convert.StartInfo.WindowStyle = ProcessWindowStyle.Hidden
          convert.StartInfo.CreateNoWindow = True
          convert.StartInfo.RedirectStandardOutput = True
          convert.StartInfo.UseShellExecute = False
          convert.StartInfo.Arguments = cmdLineArgs
          convert.StartInfo.FileName = exePath
          convert.Start()
          convert.WaitForExit()
        End Using
      End If
    End Sub
```

Итак, мы преобразовали Flash-ролик в формат MP4 или WMV. Теперь покажем, как скопировать его на устройство iTunes или Zune.

Синхронизация видеороликов с сайта YouTube с iTunes и Zune

Устройства iTunes и Zune предоставляют разные механизмы для синхронизации видео, но мы определим единый интерфейс, которым сможем воспользоваться в обоих случаях (см. примеры 4.35 и 4.36). Он позволит копировать на заданное устройство все файлы, находящиеся в указанном каталоге.

Пример 4.35. Синхронизация с iTunes и Zune (версия на C#)

```
interface IVideoService
{
  void Sync(string filePath);
}
```

Пример 4.36. Синхронизация с iTunes и Zune (версия на VB)

```
Friend Interface IVideoService
  Sub Sync(ByVal filePath As String)
End Interface
```

Синхронизация с iTunes

Для синхронизации видеороликов с iTunes мы воспользуемся библиотекой COM Interop. Она автоматически устанавливается вместе с iTunes, а для добавления ее в проект Visual Studio следует щелкнуть по нему в окне Solution Explorer и выбрать из контекстного меню пункт Add Reference.... Затем перейдите на вкладку COM и выберите из списка библиотеку типов iTunes 1.11 Type Library, которая предоставляет обертывающие COM-классы, упрощающие автоматизацию iTunes из программы для .NET.

Как следует из примеров 4.37 и 4.38, мы импортируем пространство имен `iTunesLib` и создаем закрытую переменную-член с именем `iTunes`.

Пример 4.37. Объявление класса для синхронизации с iTunes (версия на C#)

```
using iTunesLib;
public class iTunesSync : IVideoService
{
  iTunesApp iTunes = new iTunesApp();
...
```

Пример 4.38. Объявление класса для синхронизации с iTunes (версия на VB)

```
Imports iTunesLib
Public Class iTunesSync
Implements IVideoService
  Private iTunes As New iTunesApp()
...
```

Теперь нужно реализовать метод `Sync` интерфейса `IVideoService`; соответствующий код приведен в примерах 4.39 и 4.40. Для этого мы отбираем из каталога, на который указывает переменная `filePath`, только файлы с расширением MP4. Затем обходим все обнаруженные файлы и добавляем каждый в библиотеку iTunes. При вызове метода `Sync()` устройство iTunes открывается (если не было открыто ранее) и начинается добавление роликов в библиотеку. На создание миниатюры каждого видеофайла уходит от 1 до 3 секунд в зависимости от быстродействия вашего компьютера.

Пример 4.39. Реализация интерфейса IVideoService (версия на C#)

```
#region IVideoService Members
public void Sync(string filePath)
{
  // только MP4-файлы
  string[] fileList = Directory.GetFiles(filePath, "*.mp4",
    SearchOption.TopDirectoryOnly);
  try
  {
    foreach (var f in fileList)
    {
      // добавляем файл
      iTunes.LibraryPlaylist.AddFile(f);
    }
  }
  catch (Exception ex)
  {
    System.Diagnostics.Debug.WriteLine("iTunes error: " + ex.Message);
  }
  finally
  {
    this.iTunes = null;
```

```
  }
}
#endregion
```

Пример 4.40. Реализация интерфейса IVideoService (версия на VB)

```
#Region "IVideoService Members"
Public Sub Sync(ByVal filePath As String) Implements IVideoService.Sync
  ' только MP4-файлы
  Dim fileList() As String = Directory.GetFiles(filePath, "*.mp4", _
    SearchOption.TopDirectoryOnly)
  Try
    For Each f In fileList
      ' добавляем файл
      iTunes.LibraryPlaylist.AddFile(f)
    Next f
  Catch ex As Exception
    System.Diagnostics.Debug.WriteLine("iTunes error: " & ex.Message)
  Finally
    Me.iTunes = Nothing
  End Try
End Sub
#End Region
```

Примечание

Причина, по которой мы предварительно фильтруем список файлов, а не просто обходим все файлы в каталоге, заключается в том, что метод `AddFile` из iTunes COM SDK завершается с ошибкой при попытке добавить файл, не поддерживаемый iTunes (например, в формате Windows Media Video). SDK предлагает использовать метод `ConvertFile`, который преобразует файл в формат, который iTunes понимает. Хотя разработчик может воспользоваться методом `ConvertFile`, но утилита *ffmpeg.exe* поддерживает больше форматов и работать с ней из программы проще.

Как показано в примерах 4.41 и 4.42, одной строчки кода достаточно, чтобы попросить iTunes обновить устройство iPod (если оно подключено). И напоследок, поскольку мы работаем с COM, а класс iTunesApp не реализует метод `Dispose`, не забудьте освободить память, установив переменную в `null`.

Пример 4.41. Обновление iPod и очистка памяти (версия на C#)

```
public void UpdateIPod()
{
  iTunes.UpdateIPod();
}

~iTunesSync()
{
  if (this.iTunes != null)
  {
```

```
    // очистка
    this.iTunes = null;
  }
}
```

Пример 4.42. Обновление iPod и очистка памяти (версия на VB)

```
Public Sub UpdateIPod()
  iTunes.UpdateIPod()
End Sub

Protected Overrides Sub Finalize()
  If Me.iTunes IsNot Nothing Then
    ' очистка
    Me.iTunes = Nothing
  End If
End Sub
```

Синхронизация видеороликов с Zune

Процедура синхронизации видео с Zune совершенно непохожа на синхронизацию с iTunes. Хотя разработчики Zune не предоставили API, которым можно было бы воспользоваться из программы, добавление файлов в библиотеку Zune реализуется простым копированием их в каталоги, которые Zune отслеживает.

Примечание

В действительности Zune запускает в фоновом режиме процесс *ZuneLauncher.exe*. Концептуально *ZuneLauncher* реализует ту же функциональность, что и .NET-класс `FileSystemWatcher`, поскольку он следит за изменениями, которым подвергаются каталоги с аудио- и видеофайлами.

Папки, отслеживаемые Zune

Чтобы узнать, за какими папками следит Zune, нужно повозиться с реестром Windows. Точнее, мы должны открыть раздел реестра *HKEY_CURRENT_USER\Software\Microsoft\Zune\Groveler*, как показано на рис. 4.9.

Чтобы сделать то же самое программно, мы определим перечисление `ZuneMonitoredFolders`, в котором будут представлены все разделы реестра, соответствующие отслеживаемым папкам. Затем мы вызовем метод `GetZuneMonitoredFolders`, который вернет массив строк, содержащих пути к папкам. Для этого он обращается к методу `GetValue` класса `Registry`, передавая ему подраздел Groveler (примеры 4.43 и 4.44).

Пример 4.43. Программное получение из реестра списка папок, отслеживаемых Zune (версия на C#)

```
public enum ZuneMonitoredFolders
{
  MonitoredAudioFolders,
```

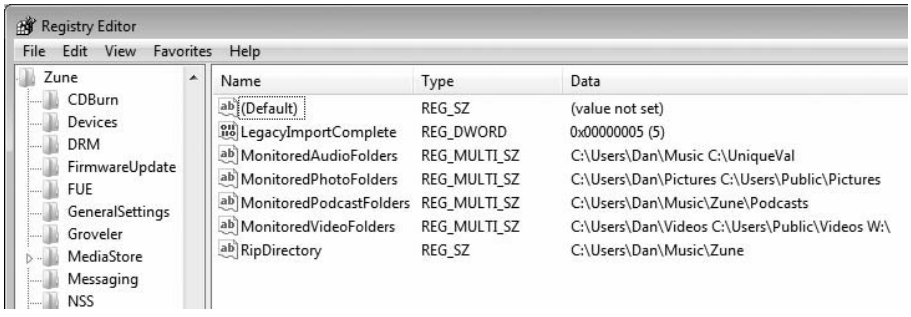

Рис. 4.9. *Папки, отслеживаемые Zune, перечислены в реестре*

```
  MonitoredVideoFolders,
  MonitoredPhotoFolders,
  MonitoredPodcastFolders
}

public class ZuneSync : IVideoService
{
  public static string[] GetZuneMonitoredFolders(ZuneMonitoredFolders
folder)
  {
    // обращаемся к реестру
    string hive = @"Software\Microsoft\Zune\Groveler\";
    string[] values =
(string[])Registry.CurrentUser.OpenSubKey(hive).GetValue(folder.ToString());
    return values;
  }
...
```

Пример 4.44. Программное получение из реестра списка папок, отслеживаемых Zune (версия на VB)

```
Public Enum ZuneMonitoredFolders
  MonitoredAudioFolders
  MonitoredVideoFolders
  MonitoredPhotoFolders
  MonitoredPodcastFolders
End Enum

Public Class ZuneSync
  Implements IVideoService

  Public Shared Function GetZuneMonitoredFolders(ByVal folder As _
    ZuneMonitoredFolders) As String()
    ' обращаемся к реестру
    Dim hive As String = "Software\Microsoft\Zune\Groveler\"
    Dim values() As String = _
    CType(My.Computer.Registry.CurrentUser.OpenSubKey(hive). _
```

```
      GetValue(folder.ToString()), String())
    Return values
  End Function
...
```

Аналогично случаю с оберткой iTunes, мы реализуем метод `IVideoService.Sync` для Zune, как показано в примерах 4.45 и 4.46. В реализации метода `Sync` мы сначала получаем список отслеживаемых папок. Если переданная в качестве параметра папка уже отслеживается, то больше делать нечего, так как помещаемые в нее видеоролики будут автоматически копироваться в Zune. Если ее нет в списке отслеживаемых папок, то мы, как и для iTunes, отфильтровываем только файлы с расширением *.wmv* и копируем их в первую отслеживаемую папку, `currentFolders[0]`.

Пример 4.45. Реализация интерфейса IVideoService (версия на C#)

```
#region VideoService Members
public void Sync(string filePath)
{
  string[] currentFolders = ZuneSync.GetZuneMonitoredFolders(
    ZuneMonitoredFolders.MonitoredVideoFolders);
  bool found = currentFolders.Contains(filePath);
  // проверяем, добавлены ли файлы в отслеживаемую папку
  if (!found)
  {
    // копируем файлы в первую отслеживаемую папку
    if (currentFolders.Length >0)
    {
      string destinationPath = currentFolders[0];
      string[] Files = Directory.GetFiles(filePath, "*.wmv",
        SearchOption.TopDirectoryOnly);
      foreach (var f in Files)
      {
        File.Copy(f, destinationPath, true);
      }
    }
    else
    {
      throw new ArgumentException("Zune is not configured to monitor *any* " +
        "folders, to fix this, open zune.exe, click settings, " +
        "and add a video folder");
    }
  }
}
#endregion
```

Пример 4.46. Реализация интерфейса IVideoService (версия на VB)

```
#Region "VideoService Members"
Public Sub Sync(ByVal filePath As String) Implements IVideoService.Sync
  Dim currentFolders() As String = ZuneSync.GetZuneMonitoredFolders( _
```

```
    ZuneMonitoredFolders.MonitoredVideoFolders)
  Dim found As Boolean = currentFolders.Contains(filePath)
  ' проверяем, добавлены ли файлы в отслеживаемую папку
  If (Not found) Then
  ' копируем файлы в первую отслеживаемую папку
    If currentFolders.Length >0 Then
      Dim destinationPath As String = currentFolders(0)
      Dim Files() As String = Directory.GetFiles(filePath, "*.wmv", _
        SearchOption.TopDirectoryOnly)
      For Each f In Files
        File.Copy(f, destinationPath, True)
      Next f
    Else
     Throw New ArgumentException("Zune is not configured to monitor *any* " & _
      "folders,to fix this, open zune.exe, click settings, and add
           a video folder")
    End If
  End If
End Sub
#End Region
```

А теперь все вместе

Итак, вы видели как:

- преобразовать XML-документы, полученные от YouTube API, в объекты;
- скачать с YouTube видеоролик и его миниатюру;
- конвертировать скачанный с YouTube видеоролик в формат WMV или MP4;
- синхронизировать видеоролики с iTunes и Zune.

Теперь мы хотим объединить эти четыре задачи и заставить их выполняться асинхронно, чтобы с приложением можно было работать, пока ролики скачиваются. Для этой цели мы воспользуемся классом `InnerTubeFeedWorker`.

Чтобы вы представляли себе объем асинхронных операций, необходимых InnerTube, рассмотрим пример набора лент, с которым InnerTube может работать.

Ролики с наивысшим рейтингом за все время существования

В этот набор входят ролики, получившие наивысший рейтинг за все время существования сайта YouTube.

Ролики с наибольшим количеством просмотров за неделю

Тоже понятно.

Таким образом, на самом верхнем уровне класс `InnerTubeFeedWorker` должен сделать следующее:

1. Отправить два HTTP-запроса YouTube API (по одному для каждой ленты).
2. Отправить 50 запросов на скачивание миниатюр (2 ленты × 25 роликов = 50 изображений).
3. Отправить 50 запросов на получение маркеров сеанса для скачивания всех роликов и еще 50 запросов на скачивание самих FLV-файлов.
4. Конвертировать все 50 роликов из формата Flash в формат Windows Media Video (WMV). Если пользователь захотел синхронизироваться с iTunes, то количество операций конвертирования возрастает до 100, так как каждый ролик придется конвертировать и в формат WMV (так хочет Zune), и в MP4 (а это каприз iTunes).
5. Синхронизировать конвертированные файлы с iTunes и Zune.

Так как трудоемкость весьма велика, InnerTube спроектирована для работы в фоновом режиме и пользуется написанной Ами Баром (Ami Bar) библиотекой `SmartThreadPool` (имеется на сайте *www.codeproject.com*), которая позволяет создавать несколько пулов потоков и управлять ими.

Чтобы собрать все вместе, взгляните на рис. 4.10, где показаны все пять вышеперечисленных задач.

- Задача Update Feed Pool обновляет все ленты. Так как соответствующие запросы сравнительны малы (3–10 Кбайт), то мы выделяем ей три потока, считая, что каждый отрабатывает достаточно быстро.
- Задача Download Video Pool скачивает миниатюры и сами видеоролики. Каждая операция занимает много времени, так как потенци-

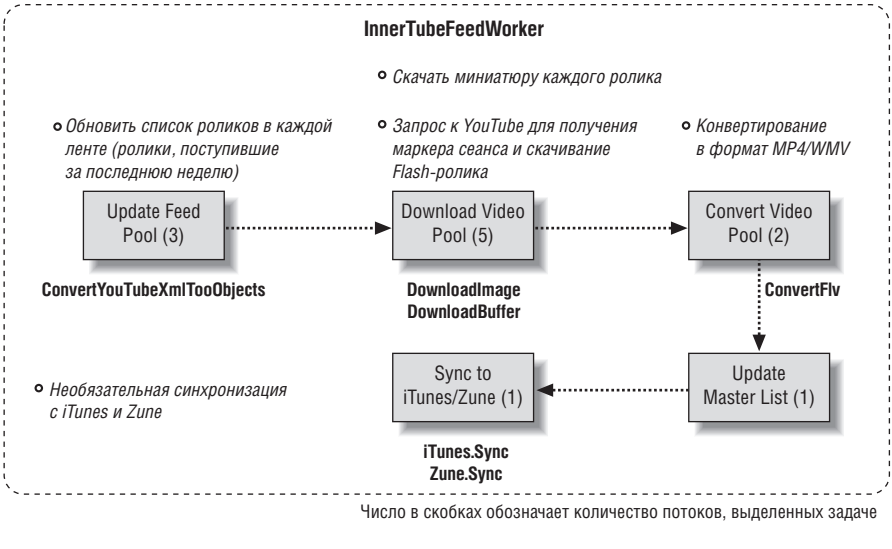

Рис. 4.10. *Задачи, выполняемые классом InnerTubeFeedWorker*

ально речь может идти о сотнях мегабайтов. Поэтому мы выделяем этой задаче пять потоков. Она нагружает главным образом сетевую и дисковую подсистему, так как скачивает большие файлы и сохраняет их на диске.

- Задача Convert Video Pool выполняется один раз после скачивания всех роликов. Она требует очень много ресурсов процессора, вы заметите, что ЦП загружен практически на 100%. Поэтому для нее мы выделим всего два потока.
- Задача Update Master List обновляет хранящийся в памяти набор роликов, который отображается в пользовательском интерфейсе приложения.
- Задача Sync to iTunes/Zune добавляет вновь конвертированные ролики в музыкальные коллекции iTunes и/или Zune.

Примечание

Вместо того чтобы переписывать статью Ами Бара с сайта CodeProject.com, где он очень подробно рассказывает о том, как работать с библиотекой, мы покажем, как вызывать фоновый объект-исполнитель `InnerTubeFeedWorker`. Полную документацию по библиотеке SmartThreadPool см. на странице *http://www.codeproject.com/KB/ threads/smartthreadpool.aspx*.

Вызов InnerTubeFeedWorker

Ниже, в коде программы, вы увидите, что для создания экземпляра класса `InnerTubeFeedWorker` мы используем не ключевое слово `new`, а статический метод `GetInstance`. Так делается для того, чтобы в любой момент времени гарантированно существовал лишь один экземпляр `InnerTubeFeedWorker`, иначе могли бы возникнуть ошибки из-за попыток загрузить и конвертировать один и тот же ролик в разных потоках, поскольку в этом случае два потока пытались бы писать в один и тот же файл, что привело бы к возникновению исключения или порче данных.

При вызове метода `GetInstance` мы передаем список объектов `InnerTubeFeeds` и настройки приложения. О классах `App` и `Setting` мы поговорим ниже в разделе «Глобальные переменные». Мы можем также задать обработчики событий, возбуждаемых объектом `InnerTubeFeedWorker` по ходу обработки и в самом конце (см. примеры 4.47 и 4.48).

Пример 4.47. Установка обработчиков событий от InnerTubeFeedWorker (версия на C#)

```
InnerTubeFeedWorker iWork =
     InnerTubeFeedWorker.GetInstance(App.InnerTubeFeeds, App.Settings);

iWork.ProgressChanged += new System.ComponentModel.
     ProgressChangedEventHandler(iWork_ProgressChanged);
iWork.RunWorkerCompleted += new System.ComponentModel.
   RunWorkerCompletedEventHandler(iWork_RunWorkerCompleted);
```

Пример 4.48. Установка обработчиков событий от InnerTubeFeedWorker (версия на VB)

```
Dim iWork As InnerTubeFeedWorker = InnerTubeFeedWorker.GetInstance( _
   App.InnerTubeFeeds, App.Settings)
AddHandler iWork.ProgressChanged, AddressOf iWork_ProgressChanged
AddHandler iWork.RunWorkerCompleted, AddressOf iWork_RunWorkerCompleted
```

Класс `InnerTubeFeedWorker` пользуется перечислением `WorkType`, которое описывает, какие именно задания он должен выполнить. Пояснения приведены в комментариях к коду в примерах 4.49 и 4.50.

Пример 4.49. Типы заданий, которые может выполнять класс InnerTubeFeedWorker (версия на C#)

```
public enum WorkType
{
  UpdateFeeds, // только обновить ленты YouTube
  Download, // только скачать FLV-файлы
  DownloadAndConvert, // скачать и конвертировать FLV-файлы
  Convert, // только конвертировать FLV-файлы
  All // все вышеперечисленное
}
```

Пример 4.50. Типы заданий, которые может выполнять класс InnerTubeFeedWorker (версия на VB)

```
Public Enum WorkType
  UpdateFeeds ' только обновить ленты YouTube
  Download ' только скачать FLV-файлы
  DownloadAndConvert ' скачать и конвертировать FLV-файлы
  Convert ' только конвертировать FLV-файлы
  All ' все вышеперечисленное
End Enum
```

Для запуска фонового исполнителя необходимо вызвать метод `RunWorkerAsync`, сообщив ему с помощью перечисления `WorkType`, что именно следует делать. Этот метод сразу же возвращает управление, а вся реальная работа происходит в отдельных фоновых потоках.

Для получения информации о текущем состоянии `InnerTubeFeedWorker` можно опросить свойство `e.UserState`, которое содержит строку с описанием текущего этапа, например: началось скачивание роликов, закончилось скачивание и т. д. (см. примеры 4.51 и 4.52). По завершении работы `InnerTubeFeedWorker` возвращает обновленный список лент, который после приведения типа можно скопировать к себе в переменную `InnerTubeFeeds`.

Пример 4.51. Получение событий, извещающих о ходе выполнения процесса (версия на C#)

```
void iWork_ProgressChanged(object sender,
  System.ComponentModel.ProgressChangedEventArgs e)
{
```

```
  string s = (string)(e.UserState);
...

void iWork_RunWorkerCompleted(object sender,
  System.ComponentModel.RunWorkerCompletedEventArgs e)
{
  App.InnerTubeFeeds = (ObservableCollection<InnerTubeFeed>)e.Result;
...
```

Пример 4.52. Получение событий, извещающих о ходе выполнения процесса (версия на VB)

```
Private Sub iWork_ProgressChanged(ByVal sender As Object, _
  ByVal e As System.ComponentModel.ProgressChangedEventArgs)
  Dim s As String = CStr(e.UserState)
...

Private Sub iWork_RunWorkerCompleted(ByVal sender As Object, _
  ByVal e As System.ComponentModel.RunWorkerCompletedEventArgs)
  App.InnerTubeFeeds = CType(e.Result, ObservableCollection(Of
InnerTubeFeed))
```

Конфигурирование WPF-приложения InnerTube

Теперь, когда с основными библиотеками мы познакомились, приступим к сборке WPF-приложения для показа списка видеороликов. Сначала необходимо определить некоторые соглашения относительно нашего приложения, а именно: где будут храниться глобальные переменные, такие как список лент и настройки приложения, содержащие информацию о том, какое клиентское ПО – iTunes или Zune – установлено и куда складывать скачанные ролики.

Глобальные переменные

Любой WPF-проект содержит класс `Application` в виде файла `App.xaml`, в котором присутствует код программной логики. Так как этот класс находится в области видимости приложения, то это лучшее место для хранения глобальных переменных. Опишем список лент и настройки приложения в качестве статических свойств, как показано в примерах 4.53 и 4.54.

Пример 4.53. Глобальные переменные InnerTube (версия на C#)

```
public partial class App : Application
{
  public static ObservableCollection<InnerTubeFeed> InnerTubeFeeds
        { get; set; }
  public static Setting Settings { get; set; }
...
```

Пример 4.54. Глобальные переменные InnerTube (версия на VB)

```
Partial Public Class App
    Inherits Application
  Private Shared privateInnerTubeFeeds As
        ObservableCollection(Of InnerTubeFeed)
  Public Shared Property InnerTubeFeeds() As
        ObservableCollection(Of InnerTubeFeed)
    Get
      Return privateInnerTubeFeeds
    End Get
    Set(ByVal value As ObservableCollection(Of InnerTubeFeed))
      privateInnerTubeFeeds = value
    End Set
  End Property

  Private Shared privateSettings As Setting
  Public Shared Property Settings() As Setting
    Get
      Return privateSettings
    End Get
    Set(ByVal value As Setting)
      privateSettings = value
    End Set
  End Property
...
```

Поскольку `InnerTubeFeeds` – глобальная переменная, в которой хранятся все ленты и ролики, то обновлять ее потребуется в следующих случаях:

- когда пользователь добавляет новую ленту в окне AddNewFeed.xaml;
- когда пользователь щелкает правой кнопкой мыши в окне *MainWindow.xaml* и выбирает из контекстного меню команду удаления ленты или ролика;
- когда пользователь запускает процесс `InnerTubeFeedWorker`, который обновляет список роликов в ленте (например, скачивает новый ролик, вошедший в избранное или получивший высокий рейтинг).

Настройки приложения

В нашем приложении есть ряд необходимых настроек, которые надо где-то хранить. Ниже перечислены поля класса `Setting` с описанием того, для чего они предназначены.

`AppName`

Имя приложения и одновременно, что важнее, имя каталога, в котором хранятся файлы приложения. По умолчанию равно `InnerTube`.

`FirstRun`

Логическое значение, используемое для определения, запускается ли приложение впервые. В этом случае мы должны показать файл

`FirstRunWindow.xaml` и создать для пользователя начальный набор лент. По умолчанию равно `true`.

`InnerTubeFeedFile`

Путь к файлу *InnerTubeFeeds.xml* на диске.

`SubPath`

Каталог, в котором хранятся миниатюры, исходные FLV-файлы и сериализованные представления таких классов, как `InnerTubeFeed` и `Setting`. Его имя совпадает со значением свойства `AppName` и по умолчанию равно `C:\Users\`*`Username`*`\Videos\InnerTube`.

`VideoPath`

Каталог, в котором хранятся конвертированные ролики. По умолчанию `C:\Users\`*`Username`*`\Videos\`.

`iTunesInstalled`

Логическое значение, равное `true`, если установлено клиентское ПО iTunes. По умолчанию равно `false`.

`ZuneInstalled`

Логическое значение, равное `true`, если установлено клиентское ПО Zune. По умолчанию равно `false`.

`UpdateFeedPoolThreads`

Целое число, равное количеству потоков в пуле задачи `UpdateFeedPool`. По умолчанию равно 3.

`DownloadPoolThreads`

Целое число, равное количеству потоков в пуле задачи `DownloadPool`. По умолчанию равно 5.

`ConversionPoolThreads`

Целое число, равное количеству потоков в пуле задачи `ConversionPool`. По умолчанию равно 2.

При первом запуске приложения код в файле *FirstRunWindow.xaml* проверяет, создан ли уже файл *Settings*. Если нет, вызывается метод `SettingService.BuildDefaultSettings`, который инициализирует вышеперечисленные поля значениями, соответствующими конкретному компьютеру.

Конструирование пользовательского интерфейса InnerTube

Пользовательский интерфейс InnerTube напоминает интерфейс Microsoft Outlook и состоит из трех панелей: слева список лент (объектов `InnerTubeFeed`), посередине – список роликов в выбранной ленте (объектов `InnerTubeVideo`), а справа – информация о выбранном ролике. В следующем разделе мы рассмотрим структуру пользовательского интер-

фейса, покажем снимок реального интерфейса и опишем необходимые элементы управления.

Структура интерфейса InnerTube

На рис. 4.11 показана структура элементов управления WPF, образующих интерфейс нашего приложения.

- На самом верхнем уровне, внутри главного окна `MainWindow`, находится элемент управления `DockPanel`, который нужен для пристыковки элементов `Menu` и `ToolBar` к верхнему краю окна.
- Для трехколонной верстки мы воспользовались элементом управления `Grid`, в котором ширина колонок задана равной *. Как и в веб-приложениях, это означает, что при изменении размеров окна колонка может расширяться.
- Первая колонка (напомним, что колонки нумеруются, начиная с 0) – это элемент `ListBox`, в котором хранится список объектов `InnerTubeFeed`, представляющих ленты.
- Вторая колонка – тоже элемент `ListBox`, но хранится там список роликов в выбранной ленте.
- Третья колонка представляет собой холст `Canvas` с несколькими элементами управления: пользовательским элементом для медиаплеера, текстовыми блоками `TextBlock` для названия ролика и т. п.

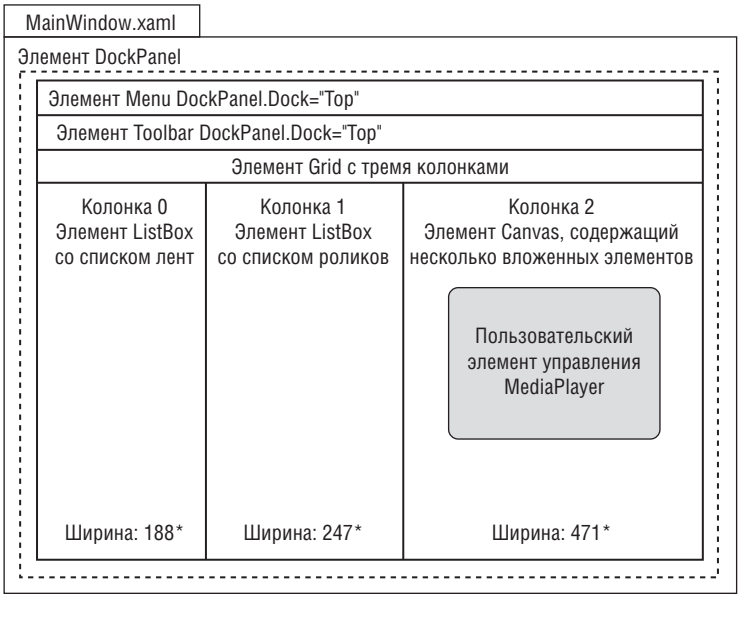

***Рис. 4.11.** Основные элементы управления в MainWindow.xaml*

Примечание

В InnerTube для определения внешнего вида кнопок, списков `ListBox` и т. п. используется файл *C4fStyle.xaml*. Описание того, что в нем находится, см. в приложении А.

Анатомия MainWindow.xaml

Поскольку сразу объяснить, как можно превратить описанную выше структуру в полнофункциональный пользовательский интерфейс, довольно трудно, разобьем файл *MainWindow.xaml* на три части, помеченные на рис. 4.12 цифрами 1, 2 и 3. Вы можете возвращаться к этому снимку, чтобы не потерять связь с описываемым пользовательским интерфейсом.

Рис. 4.12. *Сетка Grid с тремя колонками в MainWindow.xaml*

Задание контекста данных для MainWindow.xaml

Выше мы определили глобальную переменную `InnerTubeFeeds` для хранения списка лент и роликов. Чтобы привязать все элементы управления в `MainWindow` к `InnerTubeFeeds`, определим контекст данных `DataCon-`

text для этого окна в обработчике события Window_Loaded, как показано в примерах 4.55 и 4.56.

Пример 4.55. Установка контекста данных для MainWindow.xaml (версия на C#)

```
private void Window_Loaded(object sender, RoutedEventArgs e)
{
...
  this.DataContext = App.InnerTubeFeeds;
}
```

Пример 4.56. Установка контекста данных для MainWindow.xaml (версия на VB)

```
Private Sub Window_Loaded(ByVal sender As Object, ByVal e As RoutedEventArgs)
...
  Me.DataContext = App.InnerTubeFeeds
End Sub
```

Часть 1: ListBox со списком лент

Для построения списка лент мы воспользуемся шаблонным элементом управления ListBox, как показано в примере 4.57. Поскольку верхний уровень пользовательского интерфейса образован сеткой Grid, то для помещения этого элемента в первую колонку сетки необходимо присвоить его атрибуту Grid.Column значение 0 (первая колонка). Затем мы определили пункт контекстного меню ContextMenu, чтобы можно было удалить ленту (при щелчке правой кнопкой мыши). Далее мы подробнее рассмотрим вопрос о привязке к данным с помощью свойства ItemTemplate элемента ListBox.

Пример 4.57. XAML-разметка первой колонки в файле MainWindow.xaml

```
<!–Первая колонка-->
<ListBox Grid.Column="0" x:Name="feedList"
         IsSynchronizedWithCurrentItem="True"
  ItemsSource="{Binding}" Background="{x:Null}">
  <ListBox.ContextMenu>
    <ContextMenu>
      <MenuItem Click="DeleteFeed" CommandParameter="{Binding Path=/}"
        Header="Delete Feed" >
        <MenuItem.Icon>
          <Image Source="images/cross.png"></Image>
        </MenuItem.Icon>
      </MenuItem>
    </ContextMenu>
  </ListBox.ContextMenu>
  <ListBox.ItemTemplate>
    <DataTemplate>
      <StackPanel Orientation="Horizontal">
        <TextBlock FontWeight="Bold" Text="{Binding Path=FeedName}"/>
```

```
        </StackPanel>
      </DataTemplate>
    </ListBox.ItemTemplate>
  </ListBox>
```

Привязка к данным списка feedList

Для привязки списка `feedList` к данным необходимо установить несколько свойств:

`ItemSource`

Это источник данных, к которому мы привязываемся. Поскольку мы уже установили свойство `DataContext` для всего файла *MainWindow.xaml*, то это просто `"{Binding}"`.

`IsSynchronizedWithCurrentItem`

Это свойство необходимо установить в true, чтобы все элементы управления оставались синхронизированными. Это означает, что, когда пользователь щелкает по другой ленте, элемент `ListBox`, содержащий список роликов, и панель детальной информации должны быть автоматически приведены в соответствии с выбранной лентой.

`DataTemplate`

Этот шаблон управляет отображением всех связанных с данными элементов. В нашем случае мы можем привязать свойство `Text` к свойству `FeedName` объекта `App.InnerTubeFeeds`, задав такое выражение привязки: `Text="{Binding Path=FeedName}"`.

Часть 2: список VideoList

В средней колонке отображается список роликов в выбранной ленте, точнее миниатюра, название и автор ролика. Как и раньше, мы объявляем элемент управления `ListBox` и задаем для него свойство `Grid.Column`, на этот раз равное 1, так как речь идет о второй колонке. И здесь мы включаем элемент `ContextMenu`, привязывая его к клавише `Delete` путем установки свойства `InputGestureText`, а также элемент `ItemTemplate`, который определяет пользовательский интерфейс этого списка `ListBox` (см. пример 4.58).

Пример 4.58. XAML-разметка второй колонки в файле MainWindow.xaml

```
<!-Вторая колонка-->
<ListBox Grid.Column="1" IsSynchronizedWithCurrentItem="True"
  ItemsSource="{Binding Path=FeedVideos}" Name="VideoList"
  Background="{x:Null}" >
  <ListBox.ItemTemplate>
    <DataTemplate>
      <StackPanel Orientation="Horizontal">
        <StackPanel.ContextMenu>
          <ContextMenu Name="mnuDeleteVideo">
            <MenuItem Click="DeleteVideo" CommandParameter="{Binding}"
```

```
                Header="Foo" InputGestureText="Del" >
                <MenuItem.Icon>
                  <Image Source="images/cross.png"></Image>
                </MenuItem.Icon>
              </MenuItem>
            </ContextMenu>
          </StackPanel.ContextMenu>
          <Image Margin="2,2,2,2" Source="{Binding Path=DownloadedImage,
            Converter={StaticResource img} }"
            Width="48" Height="48" VerticalAlignment="Center"></Image>
          <StackPanel VerticalAlignment="Center">
            <TextBlock FontWeight="Bold" Text="{Binding Path=Title}"
              TextTrimming="WordEllipsis" TextWrapping="Wrap" />
            <TextBlock Foreground="White" Text="{Binding Path=Author}" />
          </StackPanel>
        </StackPanel>
      </DataTemplate>
    </ListBox.ItemTemplate>
  </ListBox>
```

Шаблон данных

Задержимся ненадолго, чтобы понять, каким образом XAML-разметка порождает шаблон, изображенный на рис. 4.13. Все элементы управления, определенные в шаблоне данных `DataTemplate`, находятся внутри панели `StackPanel`, для которой установлена ориентация `Horizontal`, то есть элементы располагаются в ряд слева направо.

Если смотреть слева направо, то в примере 4.59 мы сначала встречаем картинку (`Image`), для которой оба боковых поля равны 2 пикселам (независимо от устройства), а ширина и высота фиксированы. Затем идет элемент `StackPanel`, а, поскольку свойство `Orientation` для него явно не задано, то предполагается ориентация по вертикали (сверху вниз). Внутри этой панели находятся два текстовых блока `TextBlock`, представляющих название и автора ролика.

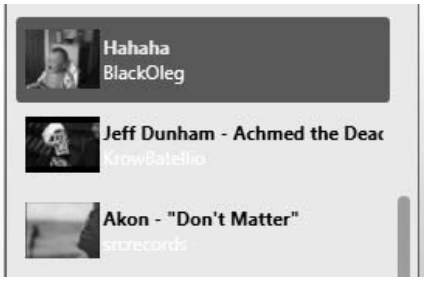

***Рис. 4.13.** Шаблон данных элемента ListBox*

Пример 4.59. XAML-разметка шаблона данных для списка роликов

```
<DataTemplate>
...
  <StackPanel Orientation="Horizontal">
    <Image Margin="2,2,2,2" Source="{Binding Path=DownloadedImage,
      Converter={StaticResource img} }"
      Width="48" Height="48" VerticalAlignment="Center">
    </Image>
    <StackPanel VerticalAlignment="Center">
      <TextBlock FontWeight="Bold" Text="{Binding Path=Title}"
        TextTrimming="WordEllipsis" TextWrapping="Wrap" />
      <TextBlock Foreground="White" Text="{Binding Path=Author}" />
    </StackPanel>
  </StackPanel>
</DataTemplate>
```

Привязка к данным элемента ListBox со списком роликов

Как и в случае списка лент, для привязки списка роликов к данным необходимо установить два свойства:

`ItemSource`

Поскольку на верхнем уровне установлена привязка к источнику данных `InnerTubeFeeds`, то нам нужно просто привязаться к свойству `FeedVideos`, поэтому установим `ItemSource` равным `"{Binding Path=FeedVideos}"`. Свойство `FeedVideos` — это набор объектов `InnerTubeVideo`, то есть мы можем присвоить `Binding Path` любое свойство класса `InnerTubeVideo`.

`IsSynchronizedWithCurrentItem`

Как и раньше, мы должны установить это свойство в true, чтобы все элементы управления в окне оставались синхронизированными.

Привязка к изображению

Взгляните на синтаксис привязки для элемента `Image` и обоих элементов `TextBlock` в примере 4.59. Если текстовые блоки просто привязываются к свойствам `Title` и `Author`, то для привязки `Image` потребовалось нечто новенькое: `Converter` и `StaticResource`. Потратим немного времени на то, чтобы объяснить назначение конвертеров и статических ресурсов.

Создание конвертера изображения

Конвертер изображения нужен потому, что мы хотим выполнить свой код на этапе привязки к данным, дабы убедиться, что картинка, указанная в свойстве `DownloadedImage`, действительно существует; если это не так, то мы подставим изображение по умолчанию. `Converter` — это любой класс .NET, реализующий интерфейс `IValueConverter`, в котором

есть всего два метода: Convert и ConvertBack. Тот и другой принимают значение типа object и возвращают результат типа object, как показано в примерах 4.60 и 4.61.

Пример 4.60. Интерфейс IValueConverter (версия на C#)

```
public interface IValueConverter
{
  object Convert(object value, Type targetType,
    object parameter, CultureInfo culture);

  object ConvertBack(object value, Type targetType,
    object parameter, CultureInfo culture);
}
```

Пример 4.61. Интерфейс IValueConverter (версия на VB)

```
Public Interface IValueConverter
  Function Convert(ByVal value As Object, ByVal targetType As Type, _
    ByVal parameter As Object, ByVal culture As CultureInfo) As Object

  Function ConvertBack(ByVal value As Object, ByVal targetType As Type, _
    ByValparameter As Object, ByVal culture As CultureInfo) As Object
End Interface
```

В следующем примере определена наша реализация интерфейса IValueConverter. Объясним, что происходит в момент привязки к данным. В соответствии с тем, как мы задали элемент Converter для источника изображения Image, WPF сначала передаст значение свойства DownloadedImage методу конвертера Image.Convert. Этот метод проверяет, что значение DownloadedImage не равно null, не пусто и что файл с таким именем действительно существует. Если все условия выполнены, то метод Convert просто возвращает путь к изображению. Если же указанной картинки не существует, вместо нее возвращается картинка-заглушка. Полный текст класса ConverterImage приведен в примерах 4.62 и 4.63.

Пример 4.62. Конвертер изображений (версия на C#)

```
[ValueConversion(typeof(object), typeof(string))]
public class ConverterImage : IValueConverter
{
public object Convert(object value, Type targetType, object parameter,
  System.Globalization.CultureInfo culture)
{
  string path = (string)value;
  return SetImage(path);
}

public static object SetImage(string imagePath)
{
  // если такой путь не существует
  if (String.IsNullOrEmpty(imagePath) || (!File.Exists(imagePath)))
  {
```

```
    // берем картинку-заглушку
    return FileHelper.DefaultImage;
  }
  else
  {
    return imagePath;
  }
}

public object ConvertBack(object value, Type targetType,
  object parameter, System.Globalization.CultureInfo culture)
  {
    // этот метод никогда не будет вызван
    return null;
  }
}
```

Пример 4.63. Конвертер изображений (версия на VB)

```
<ValueConversion(GetType(Object), GetType(String))> _
Public Class ConverterImage
  Implements IValueConverter

  Public Function Convert(ByVal value As Object, ByVal targetType As Type, _
    ByVal parameter As Object, _
    ByVal culture As System.Globalization.CultureInfo) As Object
    Implements IValueConverter.Convert

    Dim path As String = CStr(value)
      Return SetImage(path)
  End Function

  Public Shared Function SetImage(ByVal imagePath As String) As Object
    ' если такой путь не существует
    If String.IsNullOrEmpty(imagePath) OrElse ((Not
                  File.Exists(imagePath))) Then
      ' берем картинку-заглушку
      Return Path.Combine(App.Settings.SubPath, FileHelper.DefaultImage)
    Else
      Return imagePath
    End If
  End Function

  Public Function ConvertBack(ByVal value As Object,
    ByVal targetType As Type, _
    ByVal parameter As Object, _
    ByVal culture As System.Globalization.CultureInfo) As Object
    Implements IValueConverter.ConvertBack
    ' этот метод никогда не будет вызван Return Nothing
  End Function
End Class
```

Конфигурирование класса как статического ресурса

О том, как написать класс, реализующий интерфейс `IValueConverter`, мы рассказали. Теперь покажем, как этот класс можно декларативно вызвать из XAML-кода, прописав его в качестве статического ресурса (`StaticResource`) в файле *MainWindow.xaml*. Статический ресурс – это способ определения класса или другого ресурсного файла в XAML-разметке. Чтобы назначить написанный выше класс `ConverterImage` статическим ресурсом, мы должны сделать две вещи. Во-первых, требуется явно определить пространство имен, в котором находится класс `ConverterImage`, и выбрать для этого пространства имен псевдоним. Я назвал его `Util`, как показано в примере 4.64.

Пример 4.64. Объявление пространства имен .NET в XAML-разметке

```
<Window
    xmlns:Util="clr-namespace:InnerTube"
```

Во-вторых, мы определяем класс `ConverterImage` в качестве статического ресурса, включив его в словарь `ResourceDictionary`, как показано в примере ниже. Обратите внимание, что мы указываем пространство имен `InnerTube` (оно же `Util`), за которым следует имя класса, а затем определяем ключ, по которому можно ссылаться на ресурс в XAML-разметке.

Пример 4.65. Объявление класса и назначение ему ключа в XAML-разметке

```
<Window.Resources>
<ResourceDictionary>
    <!-- Конвертеры значений для привязки к данным -->
      <Util:ConverterImage x:Key="img" />
  </ResourceDictionary>
</Window.Resources>
```

Часть 3: панель детальной информации

Здесь отображается подробная информация о выбранном ролике `InnerTubeVideo`. Сама панель представлена элементом `Canvas`, для которого свойство `Grid.Column` равно 2, что соответствует третьей колонке сетки, так как нумерация колонок начинается с нуля. В элемент `Canvas` вложено несколько элементов управления, для каждого из которых заданы свойства `Canvas.Top` и `Canvas.Left`, как показано в примере 4.66.

Примечание

Опытные разработчики, использующие WPF, обычно не любят пользоваться элементом `Canvas`, так как позиции и размеры в нем задаются в пикселах, вследствие чего он не так хорошо адаптируется к изменению размера окна, как другие элементы. Но в некоторых случаях это удобно, к тому же линии сетки в Visual Studio позволяют выравнивать метки и соответствующие им значения, не прибегая к возне с XAML вручную.

Пример 4.66. XAML-разметка панели детальной информации

```
<!-- Третья колонка -->
<Canvas Grid.Column="2" Name="canvas1">
  <!-- Название -->
  <TextBlock Text="{Binding Path=FeedVideos/Title}" Name="VideoTitle"
    TextAlignment="Center" Height="28.453" Canvas.Left="18.859"
    Canvas.Top="9.228" Width="438.133" FontSize="16"
    FontWeight="Bold"></TextBlock>

  <!-- Видеоролик -->
  <Util:MediaPlayer InnerTubeVideoFile="{Binding Path=FeedVideos/}"
    x:Name="VideoPlayer" Canvas.Left="18.571" Canvas.Top="47.678"
    Height="398.342" Width="438.421" />

  <!-- Описание -->
  <ScrollViewer Canvas.Left="32.867" Canvas.Top="458.425" Height="86.684"
    Name="scrollViewer1" Width="424.413">
    <TextBlock Height="180" Text="{Binding Path=FeedVideos/Description}"
      Name="txtDescription" TextDecorations="None" TextWrapping="Wrap" />
  </ScrollViewer>

  <!-- Автор -->
  <Label Canvas.Left="46" Canvas.Top="551.777" Height="28.339"
     Name="lblAuthor" Width="120"
     HorizontalContentAlignment="Right">Author:</Label>
  <TextBlock Text="{Binding Path=FeedVideos/Author}" Canvas.Left="173"
      Canvas.Top="559.209" Height="20.907" Name="txtAuthor" Width="120" />

  <!-- Количество просмотров -->
  <Label Canvas.Left="46" Canvas.Top="580.116" Height="28.339"
     Name="lblViews" Width="120"
     HorizontalContentAlignment="Right">Views:</Label>
  <TextBlock Text="{Binding Path=FeedVideos/Views,StringFormat=N0 }"
     Name="txtViews" Canvas.Left="173" Canvas.Top="587.548"
     Height="20.907" Width="120"/>

  <!-- Средний рейтинг -->
  <Label Canvas.Left="46" Canvas.Top="610.122" Height="28.339"
     Name="lblRaters" Width="120"
     HorizontalContentAlignment="Right">Average Rating: </Label>
  <Util:RatingUserControl StarRating="{Binding Path=FeedVideos/AvgRating}"
     x:Name="Rating" Canvas.Left="173" Canvas.Top="615.123"
     Height="16.67" />

  <!-- Количество оценивших -->
  <Label Name="lblNumRating" Canvas.Left="46" Canvas.Top="636.794"
     Height="28.339" Width="120"
     HorizontalContentAlignment="Right"># of Ratings:</Label>
  <TextBlock Text="{Binding Path=FeedVideos/NumRaters, StringFormat=N0}"
     Name="txtNumRating" Canvas.Left="173" Canvas.Top="641.795"
     Height="21.671" Width="120" />
```

```
    <!-- Дата публикации -->
    <Label Canvas.Left="46" Canvas.Top="660.132" Height="28.339"
       Name="lblPublished" Width="120"
       HorizontalContentAlignment="Right">Published:</Label>
    <TextBlock Text="{Binding Path=FeedVideos/Published}" Name="txtPublished"
        Canvas.Left="173" Canvas.Top="665.133" Height="21.671" Width="216" />
</Canvas>
```

Форматирование чисел в WPF

В таких полях, как `txtViews` и `txtNumRating`, мы можем отформатировать числовые значения с помощью свойства `StringFormat`, присвоив ему допустимую в .NET форматную строку, например `N0` (число с запятыми, разделяющими группы разрядов), как показано в примере 4.67.

Пример 4.67. Привязка к данным поля «Общее число просмотров ролика» с помощью конвертера значения

```
<TextBlock Text="{Binding Path=FeedVideos/Views,StringFormat=N0 }" ...
```

Поскольку свойство `txtViews` имеет тип `int`, то при попытке привязать его к данным, не задавая `StringFormat`, мы получили бы представление вида 8675309. А после задания `StringFormat` значение `txtViews` имеет вид 8,675,309.

Привязка к данным для панели детальной информации

В отличие от двух предыдущих колонок, панель детальной информации составлена из отдельных, не связанных между собой элементов, поэтому мы не определяем свойства `ItemSource` и `IsSynchronizedProperty`. В результате синтаксис привязки каждого элемента выглядит несколько иначе. Точнее, все выражения `Binding Path` начинаются от корня, т. е. от класса `InnerTubeFeed`, следовательно, для получения названия ролика необходимо включить в путь свойство `FeedVideos`, как показано в примере 4.68.

Пример 4.68. Привязка названия ролика к данным

```
<TextBlock Text="{Binding Path=FeedVideos/Title}" ...
```

Привязка к данным пользовательских элементов управления

Возвращаясь к XAML-разметке панели детальной информации, обратите внимание, что для воспроизведения ролика мы привязываемся к пользовательскому элементу управления `MediaPlayer`, у которого имеется свойство `InnerTubeVideoFile`. Как видите, свойство `Path` установлено в `FeedVideos/`, то есть во время привязки к данным мы передаем весь выбранный в данный момент объект `InnerTubeVideo` в качестве значения свойства `InnerTubeVideoFile` (пример 4.69).

Пример 4.69. XAML-разметка для передачи объекта InnerTubeVideo пользовательскому элементу управления MediaPlayer

```
<Util:MediaPlayer InnerTubeVideoFile="{Binding Path=FeedVideos/}"
```

Пользовательский интерфейс элемента MediaPlayer

Пользовательский интерфейс элемента MediaPlayer образован сеткой `Grid`, включающей четыре элемента. Первые два – `Image` и `MediaElement` – расположены поверх друг друга (значения свойства `Margin` у них в точности совпадают), как показано в примере 4.70. Элемент `Image` будет служить для отображения миниатюры при первой загрузке, но, как только начнется воспроизведение, мы скроем этот элемент. Поскольку оба элемента занимают в точности одно и то же место, то с помощью атрибута `Grid.ZIndex` мы можем указать, что `Image`, имеющий большее значение `ZIndex`, должен располагаться поверх `MediaElement`.

Далее идет панель `StackPanel`, содержащая всего один элемент `ToggleButton`, который переключает режим между воспроизведением и паузой. Внутри него находится элемент `Path`, представляющий собой XAML-форму кнопки Play.

Пример 4.70. XAML-разметка пользовательского элемента управления MediaPlayer

```
<Grid>
  <Image Grid.ZIndex="1" Margin="5,0,5,73" Name="PreviewImage" ></Image>
  <MediaElement Grid.ZIndex="0" Margin="5,0,5,73" Name="VideoPlayer"
      LoadedBehavior="Manual"></MediaElement>
  <StackPanel Orientation="Horizontal" HorizontalAlignment="Center"
    VerticalAlignment="Bottom">
    <ToggleButton Margin="0,0,0,10" Height="50" Width="50"
        Name="PlayButton" Style="{StaticResource PlayButton}"
        Click="Play">
        <Path Data="F1M149.333,406L149.333,598 317.333,502 149.333,406z"
          Fill="#80000000" Height="14" Margin="4,0,0,0" Name="Path"
          Stretch="Fill" Width="14" />
    </ToggleButton>
  </StackPanel>
</Grid>
```

Привязка к данным элемента MediaPlayer

Для привязки элемента управления `MediaPlayer` к данным нам необходимо создать WPF-свойство специального вида, а именно *свойство зависимости*. Помимо прочих достоинств, свойства зависимости обеспечивают в WPF возможность привязки элементов управления к данным.

Определение свойства зависимости `InnerTubeVideoFile` реализуется в коде программной логики, который показан ниже. Первый шаг – определить обычное свойство, для которого метод чтения (getter) возвращает

значение типа DependencyProperty (программисты, пишущие на C#, не должны применять для этого механизм автоматических свойств, появившийся в C# 3.0), как показано в примерах 4.71 и 4.72.

Затем объявляется и регистрируется свойство зависимости, при этом указывается его имя, тип принимаемых данных, класс, для которого регистрируется свойство, и обработчик события, возникающего при изменении значения свойства. Этот обработчик заменяет метод set; так как последний никогда не вызывается, то на изменение значения можно отреагировать только в обработчике события.

Пример 4.71. Свойство зависимости InnerTubeVideoFile (версия на C#)

```
public InnerTubeVideo InnerTubeVideoFile
{
  get { return (InnerTubeVideo)GetValue(InnerTubeVideoProperty); }
  set { /* не пишите здесь код, он не будет исполняться */ }
}

public static readonly DependencyProperty InnerTubeVideoProperty =
  DependencyProperty.Register("InnerTubeVideoFile",
  typeof(InnerTubeVideo),
  typeof(MediaPlayer),
  new UIPropertyMetadata(MediaPlayer.InnerTubeVideoFileChanged));
```

Пример 4.72. Свойство зависимости InnerTubeVideoFile (версия на VB)

```
Public Property InnerTubeVideoFile() As InnerTubeVideo
  Get
    Return CType(GetValue(InnerTubeVideoProperty), InnerTubeVideo)
  End Get
  Set(ByVal value As InnerTubeVideo)
    ' не пишите здесь код, он не будет исполняться
  End Set
End Property

Public Shared ReadOnly InnerTubeVideoProperty As DependencyProperty = _
  DependencyProperty.Register("InnerTubeVideoFile", _
  GetType(InnerTubeVideo), _
  GetType(MediaPlayer), _
  New UIPropertyMetadata(AddressOf MediaPlayer.InnerTubeVideoFileChanged))
```

Теперь рассмотрим фрагмент обработчика события InnerTubeVideoFile, который будет вызываться при любом изменении свойства (примеры 4.73 и 4.74). Следует обратить внимание на два момента. Во-первых, событие объявлено статическим, поэтому обработчик не имеет доступа к членам экземпляра, представляющего текущий элемент управления MediaPlayer (конструкция this.foo в этом обработчике недопустима). Зато текущий экземпляр класса передается обработчику в виде параметра DependencyObject d, который можно привести к типу MediaPlayer и спокойно устанавливать свойства экземпляра.

Во-вторых, мы можем получить объект-владелец через параметр `DependencyPropertyChangedEventArgs e`, а затем привести его к типу `InnerTubeVideo`.

Теперь, имея текущий экземпляр и новое значение, мы можем устанавливать свойства объектов `Image` и `MediaElement`, как показано в следующих фрагментах кода.

Примечание

Элемент `MediaElement` не поддерживает воспроизведение видео в формате MP4, поэтому мы должны привязать его к WMV-файлу.

Пример 4.73. Фрагменты обработчика события, которое возникает при изменении свойства InnerTubeVideoFile (версия на C#)

```
private static void InnerTubeVideoFileChanged(DependencyObject d,
  DependencyPropertyChangedEventArgs e)
{
  MediaPlayer player = (MediaPlayer)d;
  InnerTubeVideo newVideo = (InnerTubeVideo)e.NewValue;
...
  ImageSourceConverter imageConvert = new ImageSourceConverter();
...
  player.PreviewImage.Source =
(ImageSource)imageConvert.ConvertFromString(newVideo.DownloadedImage);
...
  // установить видеофайл
  player.VideoPlayer.Source = new Uri(newVideo.DownloadedWmv);
...
```

Пример 4.74. Фрагменты обработчика события, которое возникает при изменении свойства InnerTubeVideoFile (версия на VB)

```
Private Shared Sub InnerTubeVideoFileChanged(ByVal d As DependencyObject, _
  ByVal e As DependencyPropertyChangedEventArgs)
  Dim player As MediaPlayer = CType(d, MediaPlayer)
  Dim newVideo As InnerTubeVideo = CType(e.NewValue, InnerTubeVideo)
...
  Dim imageConvert As New ImageSourceConverter()
...
  player.PreviewImage.Source = CType(imageConvert.ConvertFromString( _
    newVideo.DownloadedImage), ImageSource)
...
  ' установить видеофайл
  player.VideoPlayer.Source = New Uri(newVideo.DownloadedWmv)
...
```

Воспроизведение и приостановка ролика

Последнее, что осталось рассмотреть, – это событие `ToggleButtonClick`, объявленное, как показано в примере 4.75.

Пример 4.75. Объявление ToggleButton в XAML-разметке

```
<ToggleButton ... Name="PlayButton" Click="Play">
```

В обработчике события `Play` мы можем привести объект-отправитель к типу `ToggleButton` и посмотреть, отмечен ли сейчас флажок. В зависимости от его состояния можно делать разные вещи, например скрыть миниатюру (см. примеры 4.76 и 4.77). У элемента управления `MediaElement` имеются статические методы `Play()` и `Pause()` для воспроизведения и приостановки аудио- и видеофайлов в предположении, что источник данных задан.

Пример 4.76. Обработчик события Click элемента ToggleButton (версия на C#)

```
void Play(object sender, RoutedEventArgs args)
{
  ToggleButton tb = (ToggleButton)sender;
  if (isVideo)
  {
    // скрыть миниатюру, если это видео
    this.PreviewImage.Visibility = Visibility.Hidden;
    if ((bool)tb.IsChecked)
    {
      VideoPlayer.Play();
    }
    else
    {
      VideoPlayer.Pause();
    }
  }
}
```

Пример 4.77. Обработчик события Click элемента ToggleButton (версия на VB)

```
Private Sub Play(ByVal sender As Object, ByVal args As RoutedEventArgs)
  Dim tb As ToggleButton = CType(sender, ToggleButton)
  If isVideo Then
    ' скрыть миниатюру, если это видео
    Me.PreviewImage.Visibility = Visibility.Hidden
    If CBool(tb.IsChecked) Then
      VideoPlayer.Play()
    Else
      VideoPlayer.Pause()
    End If
  End If
End Sub
```

Заключительные замечания

В этой главе вы познакомились с тем, как работает YouTube API, как обойти ограничения на скачивание роликов, как конвертировать файлы в другой формат с помощью утилиты *ffmpeg*, как программно синхронизировать видеоролики с iTunes и Zune и как собрать все это вместе в одном приложении.

InnerTube никогда не заменит YouTube (да и задачи такой не ставилось). Но сколько раз я оказывался в ситуации, когда деваться некуда, при мне ноутбук или Zune, и можно бы убить время, вот только соединения с Интернетом нет. И тогда я с восторгом вспоминал, что на диск скачан свежий набор популярных роликов, и я могу насладиться их просмотром.

Разумеется, вы можете адаптировать приведенный код и для других сайтов, показывающих Flash-ролики, например Break.com или CollegeHumor.com, и даже придумать способ автоматического скачивания и конвертирования любого Flash-ролика в Интернете!

5

PeerCast: смотрим видео со своего компьютера из любой точки планеты

Автор	Дэн Фернандес
Сложность	Высокая
Необходимое время	10+ часов
Стоимость	Бесплатно
Программное обеспечение	Visual C# Express или Visual Basic Express, Windows Vista
Оборудование	Два ПК, один дома, другой где-то еще
Адрес в Интернете	*http://www.c4fbook.com/PeerCast*

PeerCast – это приложение на платформе Windows Presentation Foundation (WPF), которое позволяет передавать потоковое видео с одного ПК на другой по непостоянной пиринговой сети.

Краткий обзор

Когда я пытаюсь описывать, что умеет делать программа PeerCast, то поневоле уподобляюсь противно орущему продавцу информационных услуг, одетому в дешевенький костюмчик из полиэстера, испещренный вопросительными знаками. Суть PeerCast довольно проста: она передает видео с одного ПК на другой. А теперь главная приманка: если оба ПК подключены к Интернету и могут общаться между собой по протоколу Peer Name Resolution Protocol (PNRP), то транслировать видео можно из любой точки планеты. Не нужно никакого специаль-

ного ПО, не нужно вносить месячную абонентскую плату, не нужно никакого дополнительного оборудования – только ваши фильмы, транслируемые в любое время и в любое место. Позвольте мне прочистить горло и надеть свой полиэстеровый костюмчик:

> «Скучаете на работе? Хотите посмотреть записанные эпизоды из фильма «Закон и порядок» на своем Media Center, чтобы скоротать время? Теперь это возможно с программой PeerCast! А быть может, вас по работе занесло в какое-то захолустье, где из всех благ цивилизации есть только выход в Интернет (с поддержкой PNRP)? Тогда PeerCast позволит посмотреть любое видео из вашей домашней фильмотеки! И сколько, по-вашему, стоит PeerCast? Поверите, если мы скажем, что PeerCast абсолютно БЕСПЛАТНА? Поверите, если мы скажем, что отдаем вам ее исходный код даром?!! Мы спятили, раз вот так, бескорыстно, раздаем PeerCast – настолько спятили, что даже надели эти убогие полиэстеровые костюмчики, расписанные вопросительными знаками!»

Ну а теперь, отдав должное рекламной чепухе, познакомимся с возможностями PeerCast.

Экскурсия по программе PeerCast

PeerCast задумана для трансляции видео с одного ПК, выступающего в роли сервера, на другой ПК, выступающий в роли клиента, который запрашивает видео непосредственно с сервера.

Клиент или сервер?

При запуске PeerCast спрашивает, должна ли она функционировать в качестве клиента или сервера (рис. 5.1). Хотя PeerCast призвана ра-

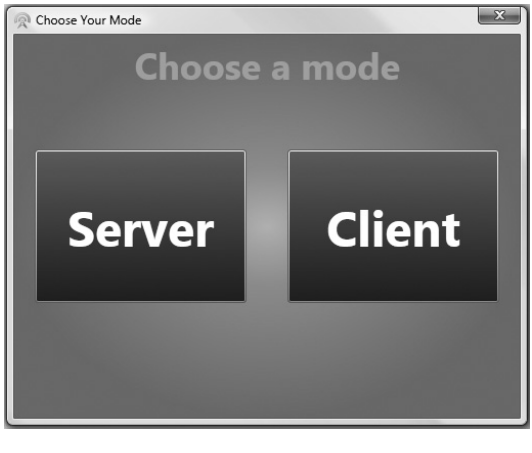

Рис. 5.1. *Выбор режима*

ботать через Интернет на нескольких ПК, вы можете одновременно запустить два ее экземпляра на одной машине, сконфигурировав один как клиент, а другой – как сервер.

Режим сервера

В режиме сервера PeerCast ждет команд от клиентов, запрашивающих трансляцию конкретного видео. В этом режиме вы можете задать местоположение видеофайлов, нажав кнопку Select. При этом открывается диалоговое окно для навигации по файловой системе, показанное на рис. 5.2.

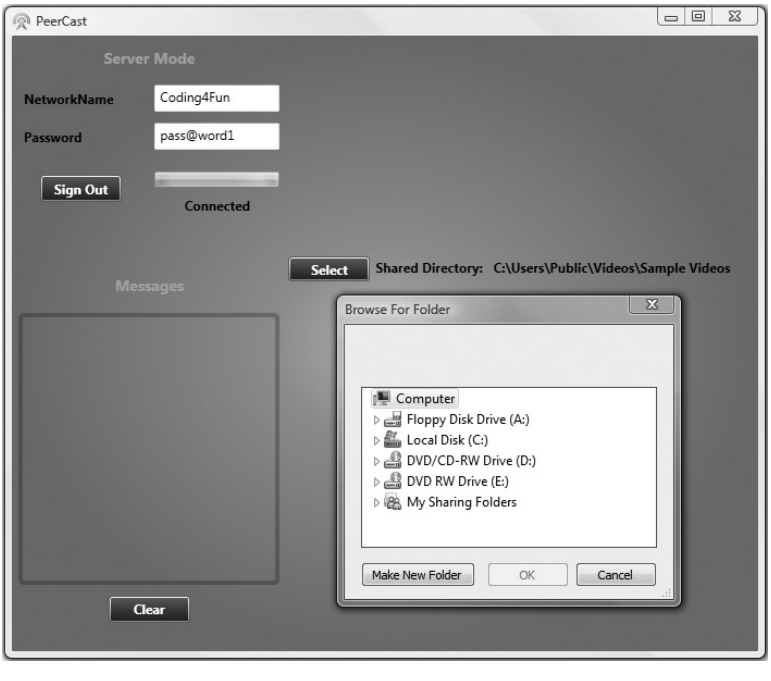

Рис. 5.2. *Запуск PeerCast в режиме сервера*

Режим клиента

В режиме клиента (рис. 5.3) мы спрашиваем, какие видеофайлы есть на сервере, и начинаем потоковую трансляцию по сети P2P.

Примечание

Работоспособность PeerCast зависит от настроек сети. Поскольку причин отказа от соединения много, мы привели советы по поиску и устранению неполадок на сайте *http://www.c4fbook.com*. Они помогут читателям диагностировать сетевые проблемы.

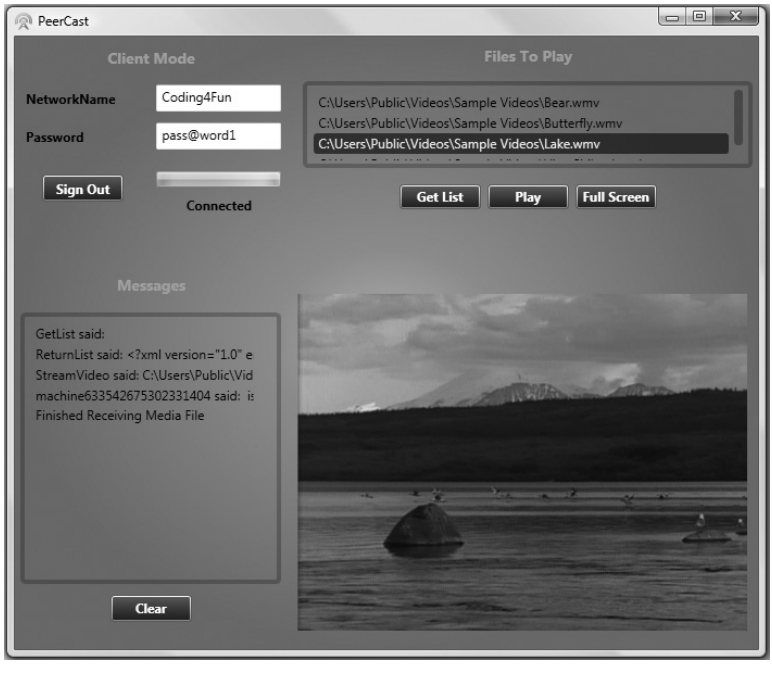

Рис. 5.3. *Трансляция видео через PeerCast*

Как работает PeerCast

Познакомившись с тем, что программа PeerCast делает, давайте рассмотрим технологию, на которой она основана, а именно:

- принципы работы протокола Peer Channel;
- построение класса `NetworkManager`;
- прослушивание P2P-событий с помощью менеджера сети (Network Manager);
- организацию процедуры входа в сеть;
- принципы работы чат-сообщений;
- принципы передачи потокового видео;
- конструирование пользовательского интерфейса *MainWindow.xaml*;
- написание кода, дополняющего *MainWindow.xaml*.

Принципы работы протокола Peer Channel

Прежде чем приступать к написанию кода, нужно объяснить, как вообще работают пиринговые приложения.

Основная терминология P2P-технологии

В этой главе не предполагается, что вы знакомы с построением P2P-приложений, поэтому мы начнем с объяснения основных терминов, употребляемых в этой области.

C4FP2P

Аббревиатура «Coding4Fun peer-to-peer», относящаяся к библиотеке, которая предоставляет обертку для протокола Peer Channel.

Ячеистая сеть (mesh)

Непостоянная (ad-hoc) сеть, составленная из узлов с одним и тем же именем ячейки (mesh name). При запуске PeerCast вы вводите имя сети и пароль и тем самым создаете ячеистую сеть. Слово «mesh» в этом контексте не имеет ни малейшего отношения к технологии Microsoft Live Mesh.

NetPeerTcpBinding

Входящий в состав каркаса Windows Communication Foundation (WCF) класс, который реализует протокол Peer Channel.

Узел (node)

Компьютер, устройство или приложение, пользующееся протоколом Peer Channel. Любой ПК, на котором работает PeerCast, можно считать узлом.

Пиринговый (Peer-to-peer – P2P)

В P2P-приложениях реализована децентрализованная коммуникация, когда клиенты обмениваются данными с такими же клиентами, а не с центральным сервером.

Peer Channel

Протокол для построения P2P-приложений, который реализует широковещательную передачу сообщений по непостоянной ячеистой сети. Программа PeerCast пользуется протоколом Peer Channel для всех пиринговых взаимодействий.

Peer Name Resolution Protocol (PNRP)

Протокол регистрации вашего узла и поиска других узлов в облаке.

Протокол Peer Channel на примере

Чтобы объяснить, как работает сеть типа Peer Channel, сравним ее с типичным клиент-серверным веб-сайтом. Когда вы заходите с помощью броузера на какой-нибудь сайт, скажем *www.msn.com*, ваш ПК незаметно для вас отправляет запрос DNS-серверу (Domain Name Server – сервер доменных имен), чтобы тот нашел IP-адрес, соответствующий доменному имени *www.msn.com*. Получив IP-адрес, броузер напрямую соединяется с соответствующим компьютером, а тот возвра-

щает ему ответ в виде HTML. Многочисленные посетители сайта общаются напрямую с сервером, а не между собой.

В сетях Peer Channel применяется протокол PNRP, который концептуально аналогичен DNS в том смысле, что оба позволяют регистрировать и разрешать символические имена, то есть сопоставлять им IP-адреса. Но в плане разрешения имен PNRP обладает рядом преимуществ по сравнению с DNS, например он позволяет регистрировать и отменять регистрацию имени в облаке PNRP в режиме реального времени, тогда как имя, зарегистрированное в системе DNS, распространяется по всему миру в течение 24-48 часов.

В отличие от клиент-серверных сетей, где есть один сервер и много клиентов, в P2P-сети коммуникации децентрализованы, то есть пиры (peers – узлы с одинаковыми обязанностями) общаются между собой в непостоянной *ячеистой сети*, составленной из узлов. Ячеистую сеть можно сравнить с комнатой в чате, которую может создать и в которую может войти любой ПК. Сама сеть защищена паролем (и, возможно, зашифрована), а входящие в нее ПК могут посылать друг другу сообщения (объекты .NET, текст, фотографии, видео), где бы они ни находились. Никакой центральный сервер для этого не нужен.

Как именно машины взаимодействуют между собой, разработчику знать необязательно, поскольку все детали берет на себя протокол Peer Channel: соединения оптимизированы, резервированы и отказоустойчивы (иными словами, если одна или несколько машин отключаются, то сеть переконфигурирует себя, так что обмен сообщениями между остальными узлами продолжается). Интересующиеся полным описанием протокола Peer Channel могут прочитать официальную спецификацию, которая находится по адресу *http://msdn.microsoft.com/en-us/library/cc219453.aspx*.

Построение пирингового приложения

Теперь, когда на концептуальном уровне вы понимаете принципы работы протокола Peer Channel, сменим пластинку и поговорим о том, как можно воспользоваться этим протоколом в нашем приложении.

Мы будем работать с библиотекой Coding4Fun P2P с открытыми исходными текстами. Она обертывает протокол Peer Channel и, в частности, включает поддержку для подключения к сети Peer Channel, отправки текстовых сообщений и трансляции потокового видео. Учитывая размер библиотеки Coding4Fun P2P, мы не станем слишком глубоко забираться в ее код, а просто объясним, как можно применить основные реализованные в ней возможности.

Асинхронное прослушивание P2P-сообщений

Одно из ключевых решений, принятых при проектировании PeerCast, заключается в том, что все сетевые коммуникации должны происхо-

дить в фоновом потоке. Это необходимо, чтобы не казалось, будто пользовательский интерфейс приложения «завис», когда принимаются P2P-сообщения или обрабатываются поступившие данные, например видеопоток.

Для обработки P2P-сообщений в фоновом потоке мы воспользуемся объектом `BackgroundWorker` и его методом `ReportProgress`, который позволяет переправлять данные из фонового потока в поток пользовательского интерфейса. Это существенно, так как в приложениях WPF применяется потоковая модель с однопотоковым апартаментом (Single-Threaded Apartment – STA), поэтому все изменения пользовательского интерфейса должны производиться только в его потоке.

Когда фоновый поток получает данные из P2P-сети, например обновленный список видеофайлов, он *не должен* напрямую изменять элементы управления в WPF-форме – при попытке сделать это будет получено исключение `InvalidOperationException`, означающее, что изменять состояние WPF-элемента может лишь тот поток, в котором этот элемент был создан. Поэтому для внесения изменений в поток пользовательского интерфейса этот поток должен быть подписан на событие `ProgressChanged`, которое возникает при каждом вызове метода `ReportProgress` в фоновом потоке. Это событие дает безопасный относительно потоков способ передать данные из фонового потока в поток пользовательского интерфейса.

Структура сообщений, передаваемых из фонового потока в поток пользовательского интерфейса

Чтобы уточнить, какие данные передаются из фонового потока в поток пользовательского интерфейса, мы явно определим типы сообщений в перечислении `UiMessage`, показанном в примерах 5.1 и 5.2.

Пример 5.1. Типы сообщений, передаваемых между потоками (версия на C#)

```
public enum UiMessage
{
  StreamVideo,
  UpdateStatus,
  ReceivedVideoList,
  ToggleUi,
  Log
}
```

Пример 5.2. Типы сообщений, передаваемых между потоками (версия на VB)

```
Public Enum UiMessage
  StreamVideo
  UpdateStatus
  ReceivedVideoList
  ToggleUi
  Log
End Enum
```

Опишем, что означает каждое сообщение:

`StreamVideo`

Сервер готов начать передачу видео.

`UpdateStatus`

Приложение подключилось к ячеистой сети или отключилось от нее.

`ReceivedVideoList`

От сервера получен XML-документ, содержащий список видеофайлов, который следует десериализовать.

`ToggleUi`

Мы вошли в систему или вышли из нее, поэтому пользовательский интерфейс необходимо перерисовать.

`Log`

По P2P-сети была отправлена команда.

Мы будем пользоваться классом `ThreadMessage`, передавая его конструктору тип сообщения (элемент перечисления `UiMessage`) и саму строку сообщения `Message`, которую нужно передать из фонового потока в поток пользовательского интерфейса (примеры 5.3 и 5.4). В классе `ThreadMessage` определен также вспомогательный метод `Create`, упрощающий создание объекта `ThreadMessage`.

Пример 5.3. Класс ThreadMessage (версия на C#)

```
public class ThreadMessage
{
  public UiMessage MessageType { get; set; }
  public string Message { get; set; }

  public static ThreadMessage Create(UiMessage messageType, string message)
  {
    return new ThreadMessage() { MessageType = messageType,
      Message = message };
  }
}
```

Пример 5.4. Класс ThreadMessage (версия на VB)

```
Public Class ThreadMessage
  Private privateMessageType As UiMessage
  Public Property MessageType() As UiMessage
    Get
      Return privateMessageType
    End Get
    Set(ByVal value As UiMessage)
      privateMessageType = value
    End Set
  End Property

  Private privateMessage As String
```

```
    Public Property Message() As String
      Get
        Return privateMessage
      End Get
      Set(ByVal value As String)
        privateMessage = value
      End Set
    End Property

    Public Shared Function Create(ByVal messageType As UiMessage, _
      ByVal message As String) As ThreadMessage
      Return New ThreadMessage() With {.MessageType = messageType,
         .Message = message}
    End Function
  End Class
```

Класс NetworkManager

Класс `NetworkManager` инкапсулирует все относящиеся к P2P функции библиотеки C4FP2P, в том числе подключение к облаку, получение текстовых сообщений, передачу потокового видео и т. д. Сама же библиотека C4FP2P обертывает класс `NetPeerTcpBinding` платформы Windows Communication Foundation.

Рассмотрим все составные части класса `NetworkManager`.

Переменные-члены и конструктор класса NetworkManager

При создании объекта `NetworkManager` инициализируется ряд переменных-членов, которые описываются ниже:

`P2pWorker`

Это объект типа `BackgroundWorker`, который создает соединение с ячеистой сетью и ожидает поступления событий. Он инициализируется в конструкторе `NetworkManager`.

`userName`

Имя пользователя, автоматически генерируемое на основе количества тиков в структуре `DateTime`. На стороне клиента и сервера это значение должно различаться.

`NetworkName`

Имя ячеистой сети, к которой мы подключаемся. Должно быть одинаково на стороне клиента и сервера.

`Password`

Пароль ячеистой сети, к которой мы подключаемся. Тоже должно быть одинаково на стороне клиента и сервера.

`p2p`

Это библиотека C4FP2P, обрабатывающая все коммуникации по протоколу Peer Channel, в том числе процедуру установления со-

единения, отправку и получение сообщений и трансляцию потокового видео.

`VideoHttpListener`

Сервер пользуется этим объектом, чтобы задать метод разбиения видеофайла на пакеты, представленные объектами `StreamPacket`. На стороне клиента при получении от сервера пакета `StreamPacket` мы вызываем метод `StreamChanged` объекта `videohttpListener`, чтобы записать `StreamPacket` в URL localhost, то есть на клиентский компьютер.

`ProgressBarValue`

Нужен для установки значения индикатора хода процесса в окне `MainWindow`, чтобы пользователь знал, насколько далеко продвинулась обработка данных. Инициализируется значением 0.

CurrentState

Элемент перечисления `SystemState`, значения которого – перечисление возможных состояний программы: вошла в систему, вышла из системы или находится в процессе входа.

Перечисление `SystemState` определено в библиотеке C4FP2P и описывает три возможных состояния нашего класса `NetworkManager`. Оно показано в примерах 5.5 и 5.6.

Пример 5.5. Перечисление SystemState (версия на C#)

```
public enum SystemState
{
  LoggedOut,
  LoggingIn,
  LoggedIn,
}
```

Пример 5.6. Перечисление SystemState (версия на VB)

```
Public Enum SystemState
  LoggedOut
  LoggingIn
  LoggedIn
End Enum
```

Переменные-члены и конструктор класса `NetworkManager` приведены в примерах 5.7 и 5.8.

Пример 5.7. Переменные-члены и конструктор класса NetworkManager (версия на C#)

```
using System;
using System.ComponentModel;
using System.Windows;
using C4F.VistaP2P.Common;

public class NetworkManager
{
```

```
  #region Variables
  public BackgroundWorker P2pWorker;
  private string userName = "machine" + DateTime.Now.Ticks;
  private string networkName;
  private string password;
  private P2PLib p2p;
  private StreamingHttpListener videoHttpListener;
  private int progressBarValue = 0;
  public SystemState CurrentState = SystemState.LoggedOut;
  #endregion

  public NetworkManager()
  {
    P2pWorker = BackgroundWorkerUtility.Create();
  }
...
```

Пример 5.8. Переменные-члены и конструктор класса NetworkManager (версия на VB)

```
Imports Microsoft.VisualBasic
Imports System
Imports System.ComponentModel
Imports System.Windows
Imports C4F.VistaP2P.Common

Public Class NetworkManager
  #Region "Variables"
  Public P2pWorker As BackgroundWorker
  Private userName As String = "machine" & DateTime.Now.Ticks
  Private networkName As String
  Private password As String
  Private p2p As P2PLib
  Private videoHttpListener As StreamingHttpListener
  Private progressBarValue As Integer = 0
  Public CurrentState As SystemState = SystemState.LoggedOut
  #End Region

  Public Sub New()
    P2pWorker = BackgroundWorkerUtility.Create()
  End Sub
```

Конструктор `NetworkManager`**, показанный в примерах 5.7 и 5.8, вызывает метод** `Create()` **класса** `BackgroundWorkerUtility`**, который создает объект типа** `BackgroundWorker`**, разрешает ему извещать главный поток о текущем состоянии и активирует возможность отменить фоновый поток-исполнитель (см. примеры 5.9 и 5.10).**

Пример 5.9. Класс BackgroundWorkerUtility (версия на C#)

```
public static class BackgroundWorkerUtility
{
  public static BackgroundWorker Create()
  {
```

```
    BackgroundWorker worker = new BackgroundWorker()
    {
      WorkerReportsProgress = true,
      WorkerSupportsCancellation = true
    };
    return worker;
  }
}
```

Пример 5.10. Класс BackgroundWorkerUtility (версия на VB)

```
Public NotInheritable Class BackgroundWorkerUtility
  Public Shared Function Create() As BackgroundWorker
    Dim worker As New BackgroundWorker() With { _
       .WorkerReportsProgress = True, _
       .WorkerSupportsCancellation = True}
    Return worker
  End Function
End Class
```

Установление соединения

Для подключения к ячеистой сети вызывается метод `PeerChannelWrapperStart` объекта P2PLib, который принимает имя узла `nodeName`, имя сети `networkName` и пароль `password`, как показано в примерах 5.11 и 5.12.

Пример 5.11. Установление P2P-соединения (версия на C#)

```
public bool PeerChannelWrapperStart(String nodeName, String networkName, _
  String password)
```

Пример 5.12. Установление P2P-соединения (версия на VB)

```
public Boolean PeerChannelWrapperStart(String nodeName, String networkName, _
  String password)
```

Если передать в качестве параметров *«MyHomePC»*, *«Coding4Fun»* и *«myPassword»*, то после регистрации имени *«MyHomePC»* в PNRP будет создана новая ячеистая сеть с именем *«Coding4Fun»* (если такой сети еще не существует) и ей будет назначен пароль *«myPassword»*.

Примечание

При запуске PeerCast имя сети и пароль на клиенте и сервере должны в точности совпадать.

Мы обернем этот вызов методом `StartConnection` класса `NetworkManager`; при этом клиент должен будет передать только `networkName` и `password`, поскольку `userName` генерируется динамически (см. примеры 5.7 и 5.8). Внутри `StartConnection` мы обращаемся к методу `ValidateResetConnection()`, который проверяет, что объект P2PLib был надлежащим образом закрыт, а затем изменяем `CurrentState`, отражая тот факт, что мы находимся в процессе установления соединения. Далее мы указываем,

что в фоновом потоке будет выполняться метод P2pWorkerAsync, и запускаем фоновый поток методом RunWorkerAsync. В результате метод P2pWorkerAsync начнет выполняться, но не в потоке пользовательского интерфейса, из которого был вызван метод StartConnection, а в фоновом потоке.

Полный текст метода StartConnection приведен в примерах 5.13 и 5.14.

Пример 5.13. Установление P2P-соединения (версия на C#)

```
public void StartConnection(string networkName, string password)
{
  this.networkName = networkName;
  this.password = password;

  if (ValidateResetConnection())
  {
    // изменить состояние подключения
    this.CurrentState = SystemState.LoggingIn;
    P2pWorker.DoWork += new
      System.ComponentModel.DoWorkEventHandler(P2pWorkerAsync);

    // начать работу
    P2pWorker.RunWorkerAsync();
  }
}
```

Пример 5.14. Установление P2P-соединения (версия на VB)

```
Public Sub StartConnection(ByVal networkName As String, ByVal password
     As String)
  Me.networkName = networkName
  Me.password = password

  If ValidateResetConnection() Then
    ' изменить состояние подключения
    Me.CurrentState = SystemState.LoggingIn

    AddHandler P2pWorker.DoWork, AddressOf P2pWorkerAsync

    ' начать работу
    P2pWorker.RunWorkerAsync()
  End If
End Sub
```

Метод ValidateResetConnection, вызываемый из метода StartConnection, просто проверяет, равна ли переменная p2p null (см. примеры 5.15 и 5.16). Хотя сразу после запуска приложения p2p равна null, а при разрыве соединения в методе EndConnection тоже сбрасывается в null (см. ниже раздел «Разрыв соединения»), существует небольшой шанс, что объект p2p не будет надлежащим образом освобожден, и тогда программа станет работать неправильно. Поэтому мы и выполняем дополнительную проверку.

Пример 5.15. Проверка корректного закрытия P2PLib (версия на C#)

```
// возможно, что после сброса p2pConnection в null
// соединение не "умерло"; проверим это еще раз здесь
private bool ValidateResetConnection()
{
  if (p2p != null)
  {
    MessageBox.Show("Ошибка при инициализации сети." +
      "Необходимо перезапустить приложение");
    return false;
  }
  return true;
}
```

Пример 5.16. Проверка корректного закрытия P2PLib (версия на VB)

```
' возможно, что после сброса p2pConnection в null
' соединение не "умерло"; проверим это еще раз здесь
Private Function ValidateResetConnection() As Boolean
  If p2p IsNot Nothing Then
    MessageBox.Show("Ошибка при инициализации сети." & _
      "Необходимо перезапустить приложение")
    Return False
  End If
  Return True
End Function
```

Запуск фонового потока

После того как метод `StartConnection` вызвал `RunWorkAsync()`, в фоновом потоке сразу же начинает работать метод `P2pWorkerAsync`, показанный в примерах 5.17 и 5.18. В нем мы вызываем метод `ReportProgress` и отправляем сообщение `ThreadMessage`, в котором поле `ClientMessage` равно «ToggleUi». Тем самым мы извещаем поток пользовательского интерфейса о том, что состояние `SystemState` изменилось и программа сейчас занята входом в сеть.

Далее мы создаем новый экземпляр класса `P2Plib` и вызываем метод `PeerChannelWrapperStart`, передавая ему имя пользователя, имя сети и пароль. Если при установлении соединения возникнет ошибка, то этот метод вернет false, в противном случае – true. Возврат true еще не означает, что мы вошли в сеть, утверждается лишь, что запрос на установление соединения успешно принят.

Поскольку вся процедура регистрации ПК и создания непостоянной сети занимает в среднем 10–30 секунд, то мы подписываемся на событие `StatusChanged` из класса `P2PLib`, чтобы получить извещение об успешном подключении. Подробнее об обработке событий `StatusChanged` мы поговорим ниже (примеры 5.25 и 5.26).

Пример 5.17. Подключение к P2P-сети (версия на C#)

```
private void P2pWorkerAsync(object sender, DoWorkEventArgs doWorkArgs)
{
  // отправляем сообщение о том, что мы входим в сеть
  P2pWorker.ReportProgress(0, ThreadMessage.Create(UiMessage.ToggleUi,
    string.Empty));

  // создаем новый объект p2p
  p2p = new P2PLib();

  // подключаемся к PeerChannel
  bool start = p2p.PeerChannelWrapperStart(userName, networkName, password);

  // проверяем ошибки подключения
  if (!start)
  {
    EndConnection();
  }

  SubscribeToPeerChannelEvents();

  // Соединение с PeerChannel может занять некоторое
  // время, поэтому запускаем индикатор ProgressBar
  CycleProgressBarUntilConnected();

  SitAndWait();
}
```

Пример 5.18. Подключение к P2P-сети (версия на VB)

```
Private Sub P2pWorkerAsync(ByVal sender As Object, ByVal doWorkArgs As _
  DoWorkEventArgs)
  ' отправляем сообщение о том, что мы входим в сеть
  P2pWorker.ReportProgress(0, ThreadMessage.Create(UiMessage.ToggleUi,
String.Empty))

  ' создаем новый объект p2p
  p2p = New P2PLib()

  ' подключаемся к PeerChannel
  Dim start As Boolean = p2p.PeerChannelWrapperStart(userName,
      networkName, password)

  ' проверяем ошибки подключения
  If (Not start) Then
    EndConnection()
  End If

  SubscribeToPeerChannelEvents()

  ' Соединение с PeerChannel может занять некоторое
  ' время, поэтому запускаем индикатор ProgressBar
  CycleProgressBarUntilConnected()
  SitAndWait()
End Sub
```

Подписка на события P2PLib

В библиотеке C4FP2P для передачи данных по P2P-сети используются события. В программе PeerCast мы подписываемся на события StatusChanged, ChatChanged и, если работаем в режиме клиента, то также на событие StreamChanged (примеры 5.19 и 5.21). Ниже описаны все три события:

StatusChanged

Возникает, когда кто-то присоединяется к ячеистой сети или покидает ее.

ChatChanged

Возникает при отправке по сети любого чат-сообщения.

StreamChanged

Возникает при получении пакета StreamPacket (фрагмента передаваемого видеофайла).

Хотя подробный разговор о классе StreamingHttpListener мы отложим до раздела «Конфигурирование видеопотока», напомним все же, что именно в этом методе мы создаем объект videoHttpListener.

Пример 5.19. Подписка на события Peer Channel (версия на C#)

```
private void SubscribeToPeerChannelEvents()
{
  // подписываемся на интересующие нас события
  p2p.StatusChanged += new
    EventHandler<StatusChangedEventArgs>(P2PLib_StatusChanged);
  p2p.TextPeerChannelHelper.ChatChanged += new
    EventHandler<ChatChangedEventArgs>(P2PLib_ChatChanged);

  // слушатель конфигурируется только в режиме клиента
  if (!App.IsServerMode)
  {
    p2p.StreamedVideo.StreamChanged += new
    EventHandler<StreamedChangedEventArgs>(P2PLib_StreamChanged);
  }

  // необходим серверу для начала трансляции и отправки очередного
  // пакета, клиенту - для события StreamChanged, определенного выше
  videoHttpListener = new StreamingHttpListener(userName + "/video/",
    MediaFinished,
    LogMessage,
    p2p.StreamedVideo.StartStream,
    p2p.StreamedVideo.SendStream);
}
```

Пример 5.20. Подписка на события Peer Channel (версия на VB)

```
Private Sub SubscribeToPeerChannelEvents()
  ' подписываемся на интересующие нас события
  AddHandler p2p.StatusChanged, AddressOf P2PLib_StatusChanged
```

```
  AddHandler p2p.TextPeerChannelHelper.ChatChanged, _
      AddressOf P2PLib_ChatChanged

  ' слушатель конфигурируется только в режиме клиента
  If (Not App.IsServerMode) Then
    AddHandler p2p.StreamedVideo.StreamChanged, AddressOf P2PLib_StreamChanged
  End If

  ' необходим серверу для начала трансляции и отправки
  ' очередного пакета, клиенту - для события StreamChanged,
  ' определенного выше
  videoHttpListener = New StreamingHttpListener(userName & "/video/", _
    AddressOf MediaFinished, AddressOf LogMessage, _
    AddressOf p2p.StreamedVideo.StartStream, _
    AddressOf p2p.StreamedVideo.SendStream)
End Sub
```

Извещение потока пользовательского интерфейса о ходе ожидания подключения

Подписавшись на события P2PLib, мы должны дождаться подключения к сети, только потом можно будет сделать что-то полезное. При возникновении события `StatusChanged`, показывающего, что соединение установлено, значение `SystemState` в объекте `NetworkManager` изменится на `SystemState.LoggedIn`. Но пока этого не произошло, фоновый поток должен поддерживать обратную связь с потоком пользовательского интерфейса.

Для этого мы воспользуемся методом `CycleProgressBarUntilConnected`, который проверяет состояние `SystemState`. Если приложение все еще занято входом в сеть, то значение переменной `progressBarValue` увеличивается на 1 и отправляется в поток пользовательского интерфейса методом `ReportProgress`, как показано в примерах 5.21 и 5.22. Когда `progressBarValue` достигает 100, мы сбрасываем значение в 0, при этом индикатор вернется в начало и снова начнет увеличиваться от 0 до 100%. Чтобы при этом не загружать ЦП на 100%, в цикле вызывается метод `Thread.Sleep`, который приостанавливает поток на 100 миллисекунд. Мы выходим из цикла, когда `SystemState` станет равно `LoggedOff` или `LoggedIn`.

Пример 5.21. Увеличение progressBarValue (версия на C#)

```
private void CycleProgressBarUntilConnected()
{
  // ждем, пока вход в сеть не завершится, обновляя при
  // этом индикатор хода процесса
  while (this.CurrentState == SystemState.LoggingIn)
  {
    System.Threading.Thread.Sleep(100);
    if (progressBarValue >= 100)
    {
      progressBarValue = 0;
```

```
    }
    else
    {
      progressBarValue++;
    }

    // отправляем сообщение в поток пользовательского интерфейса
    P2pWorker.ReportProgress(progressBarValue);
    if (P2pWorker.CancellationPending)
    {
      P2pWorker.ReportProgress(0);
    }
  }

  // по завершении отправляем 100 в качестве значения индикатора
  P2pWorker.ReportProgress(100);
}
```

Пример 5.22. Увеличение progressBarValue (версия на VB)

```
Private Sub CycleProgressBarUntilConnected()
  ' ждем, пока вход в сеть не завершится, обновляя при
  ' этом индикатор хода процесса
  Do While Me.CurrentState = SystemState.LoggingIn
    System.Threading.Thread.Sleep(100)
    If progressBarValue >= 100 Then
      progressBarValue = 0
    Else
      progressBarValue += 1
    End If

    ' отправляем сообщение в поток пользовательского интерфейса
    P2pWorker.ReportProgress(progressBarValue)
    If P2pWorker.CancellationPending Then
      P2pWorker.ReportProgress(0)
    End If
  Loop

  ' по завершении отправляем 100 в качестве значения индикатора
  P2pWorker.ReportProgress(100)
End Sub
```

Предотвращение завершения потока BackgroundWorker

После того как состояние стало равно `LoggedIn`, цикл в методе `CycleProgressBarUntilConnected` завершается, а затем вызывается метод `SitAndWait`. Внутри этого метода находится бесконечный цикл, который завершается только по получении сообщения `CancellationPending` (примеры 5.23 и 5.24). Так как мы установили обработчики событий P2PLib в фоновом потоке, то и сами события будут возбуждаться в фоновом потоке.

Пример 5.23. Предотвращение завершения фонового потока (версия на C#)

```
private void SitAndWait()
{
  // бесконечный цикл, ожидающий событий от PeerChannel или формы
  while (true)
  {
    System.Threading.Thread.Sleep(1000);
    if (P2pWorker != null)
    {
      if (P2pWorker.CancellationPending || CurrentState ==
          SystemState.LoggedOut)
      {
        P2pWorker.ReportProgress(0);
        break;
      }
    }
  }
}
```

Пример 5.24. Предотвращение завершения фонового потока (версия на VB)

```
Private Sub SitAndWait()
  ' бесконечный цикл, ожидающий событий от PeerChannel или формы
  Do
    System.Threading.Thread.Sleep(1000)
    If P2pWorker IsNot Nothing Then
      If P2pWorker.CancellationPending OrElse CurrentState = _
         SystemState.LoggedOut
      Then
        P2pWorker.ReportProgress(0)
        Exit Do
      End If
    End If
  Loop
End Sub
```

Прослушивание P2P-событий с помощью менеджера сети NetworkManager

Итак, фоновый поток `P2pWorker` продолжает работать (пока мы явно не отменим его или не выйдем из сети), и программа ждет событий от класса `P2Plib`, на которые мы подписались. Рассмотрим, как эти три события обрабатываются.

Как работает процедура входа в сеть

Прежде чем вплотную приступить к обработчику события `StatusChanged`, вспомним, что мы уже сделали, и поясним, как работает процедура входа в сеть (рис. 5.4).

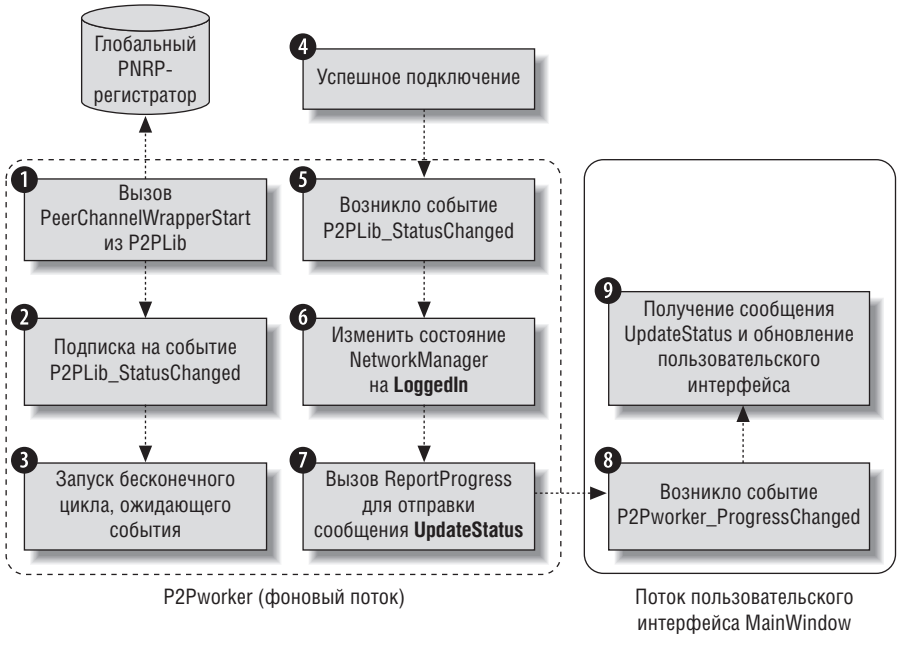

***Рис. 5.4.** Вход в P2P-сеть*

1. Метод `PeerChannelWrapperStart` начинает процедуру установления соединения, обращаясь к библиотеке C4FP2P, которая вызывает глобальный PNRP-регистратор для регистрации вашего узла (см. примеры 5.17 и 5.18).
2. Чтобы получать извещения об изменении состояния соединения, необходимо подписаться на событие `StatusChanged` (см. примеры 5.19 и 5.20).
3. Мы оставляем фоновый поток `P2Pworker` крутиться в бесконечном цикле, чтобы события P2P возникали именно в фоновом потоке, а не приводили к кажущемуся «зависанию» пользовательского интерфейса (см. примеры 5.23 и 5.24).
4. В предположении, что при подключении к глобальному облаку не возникло ошибок (к примеру, из-за старого маршрутизатора, который не поддерживает спецификацию UPnP), ваш узел будет зарегистрирован в глобальной PNRP-сети по его IPv6-адресу.
5. После регистрации узла библиотека C4FP2P получит извещение и возбудит событие `StatusChanged`, говорящее о том, что узел присоединился к сети или покинул ее (см. примеры 5.25 и 5.26).
6. Зная, что мы вошли в сеть, изменяем состояние `SystemState` менеджера сети `NetworkManager` на `LoggedIn` (см. примеры 5.25 и 5.26).

7. Чтобы известить поток пользовательского интерфейса об успешном входе в сеть, мы вызываем метод `ReportProgress`, который отправит сообщение `ThreadMessage` типа `UpdateStatus`.
8. В потоке пользовательского интерфейса возникает событие `ProgressChanged`, содержащее сообщение типа `UpdateStatus`, отправленное фоновым потоком `P2PWorker`.
9. Поток пользовательского интерфейса получает объект `ThreadMessage` и обновляет поля в методе `ReceiveUpdateStatus`, который проверяет установленное на шаге 6 состояние `SystemState` менеджера сети, чтобы понять, состоялся вход в сеть или нет (см. примеры 5.27 и 5.28).

Событие StatusChanged

Событие `StatusChanged` возникает при изменении состояния любого узла, например когда он присоединяется к сети или покидает ее. Объект `StatusChangedEventArgs` позволяет узнать, какой пользователь стал причиной события (`e.Member`), и проверить, означает ли новое состояние, что вы вошли в сеть (`e.NewNodeJoined`).

В нашем обработчике события мы сначала проверяем, указано ли в аргументе события `StatusChanged` именно наше имя. Если да, то проверяется, что произошло: вход или выход из P2P-сети, после чего вызывается метод `ReportProgress`, который извещает поток пользовательского интерфейса о том, что состоялся вход (при этом значение `progressBar` делается равным 100) или выход (тогда `progressBar` сбрасывается в 0), как показано в примерах 5.25 и 5.26.

Пример 5.25. Проверка причины события: вход или выход из сети (версия на C#)

```
protected void P2PLib_StatusChanged(object sender, StatusChangedEventArgs e)
{
  if (userName == e.Member)
  {
    if (e.NewNodeJoined)
    {
      this.CurrentState = SystemState.LoggedIn;
      P2pWorker.ReportProgress(100,
            ThreadMessage.Create(UiMessage.UpdateStatus,
        "Подключен"));
    }
    if (e.NodeLeft)
    {
      P2pWorker.ReportProgress(0,
            ThreadMessage.Create(UiMessage.UpdateStatus,
        "Отключен..."));
      this.CurrentState = SystemState.LoggedOut;
    }
  }
}
```

Пример 5.26. Проверка причины события: вход или выход из сети (версия на VB)

```
Protected Sub P2PLib_StatusChanged(ByVal sender As Object, _
  ByVal e As StatusChangedEventArgs)
  If userName = e.Member Then
    If e.NewNodeJoined Then
      Me.CurrentState = SystemState.LoggedIn
      P2pWorker.ReportProgress(100, _
            ThreadMessage.Create(UiMessage.UpdateStatus, _
            "Подключен"))
    End If
    If e.NodeLeft Then
      P2pWorker.ReportProgress(0,
            ThreadMessage.Create(UiMessage.UpdateStatus, _
        "Отключен..."))
      Me.CurrentState = SystemState.LoggedOut
    End If
  End If
End Sub
```

Отправка команд с помощью чат-сообщений

Программа PeerCast должна отправлять команды от клиента к серверу и обратно, например чтобы известить сервер о желании клиента начать получение видео. Для отправки команд мы воспользуемся имеющимся в P2PLib механизмом передачи текстовых сообщений. В примерах 5.27 и 5.28 определено перечисление `ChatCommand`, в котором описаны все команды, используемые в нашем приложении.

Пример 5.27. Перечисление ChatCommand (версия на C#)

```
public enum ChatCommand
{
  GetList,
  ReturnList,
  StreamVideo
}
```

Пример 5.28. Перечисление ChatCommand (версия на VB)

```
Public Enum ChatCommand
  GetList
  ReturnList
  StreamVideo
End Enum
```

Ниже приведены краткие описания всех типов чат-команд:

`GetList`

Клиент запрашивает список видеофайлов, которыми располагает сервер.

`ReturnList`

Сервер отправляет клиенту список видеофайлов.

`StreamedVideo`

Клиент просит сервер начать передачу конкретного видеофайла.

Аналогично тому, как мы создали класс `ThreadMessage`, с помощью которого мы передавали сообщения из фонового потока в поток пользовательского интерфейса, создадим класс `ChatMessage`, который будет представлять сообщения, которыми обмениваются узлы сети.

В классе `ChatMessage` есть два свойства: `ChatCommand`, о котором мы только что рассказали, и `Message` (см. примеры 5.29 and 5.30). Имеется также вспомогательный метод `Create`, упрощающий создание объектов типа `ChatMessage`.

Пример 5.29. Класс ChatMessage (версия на C#)

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

public class ChatMessage
{
  public ChatCommand CommandType { get; set; }
  public string Message { get; set; }

  public static ChatMessage Create(ChatCommand command, string message)
  {
    return new ChatMessage() { CommandType = command, Message = message };
  }
...
```

Пример 5.30. Класс ChatMessage (версия на VB)

```
Public Class ChatMessage
  Private privateCommandType As ChatCommand
  Public Property CommandType() As ChatCommand
    Get
      Return privateCommandType
    End Get
    Set(ByVal value As ChatCommand)
      privateCommandType = value
    End Set
  End Property

  Private privateMessage As String
  Public Property Message() As String
    Get
      Return privateMessage
    End Get
    Set(ByVal value As String)
      privateMessage = value
```

```
    End Set
  End Property

  Public Shared Function Create(ByVal command As ChatCommand, _
    ByVal message As String) As ChatMessage
    Return New ChatMessage() With {.CommandType = command, .Message = message}
  End Function
```

Для отправки сообщения по сети в классе `NetworkManager` есть метод `SendChatMessage`, который принимает объект `ChatMessage` и переадресует запрос методу `SendTextMessage` из библиотеки P2PLib. Метод `SendTextMessage` принимает два параметра: отправитель и сообщение. В качестве отправителя мы задаем значение `CommandType.ToString()`, то есть строковое представление типа сообщения, а в качестве сообщения – значение `ChatMessage.Message`, как показано в примерах 5.31 и 5.32.

Пример 5.31. Отправка ChatMessage (версия на C#)

```
public void SendChatMessage(ChatMessage message)
{
  p2p.TextPeerChannelHelper.SendTextMessage(message.CommandType.ToString(),
    message.Message);
}
```

Пример 5.32. Отправка ChatMessage (версия на VB)

```
Public Sub SendChatMessage(ByVal message As ChatMessage)
  p2p.TextPeerChannelHelper.SendTextMessage(message.CommandType.ToString(), _
    message.Message)
End Sub
```

В реализации класса `TextPeerChannelHelper` из библиотеки P2PLib есть один подвох: содержимое сообщения, отправляемого методом `SendTextMessage`, модифицируется при передаче по сети!

Пусть, например, вы хотите отправить строку `StreamVideo` (результат вызова метода `ToString()` для значения `ChatCommand.StreamVideo`), чтобы начать трансляцию файла *Bear.wmv* (см. примеры 5.33 и 5.34).

Пример 5.33. Отправка чат-сообщения с помощью P2PLib (версия на C#)

```
SendTextMessage("StreamVideo", @"C:\Users\Public\Bear.wmv");
```

Пример 5.34. Отправка чат-сообщения с помощью P2PLib (версия на VB)

```
SendTextMessage("StreamVideo", "C:\Users\Public\Bear.wmv")
```

Мы ожидаем, что будет получено что-то вроде:

```
"StreamVideo C:\Users\Public\Bear.wmv"
```

но библиотека P2PLib вставляет строку `" said: "` между командой и сообщением (включая по одному пробелу слева и справа), поэтому получатель видит следующее:

```
"StreamVideo said: C:\Users\Public\Bear.wmv"
```

Для удаления лишнего текста нам понадобится вспомогательный метод `P2PLibParse` в классе `ChatMessage`, который выбрасывает строку `" said: "` и преобразует остаток в объект типа `ChatMessage`, как показано в примерах 5.35 и 5.36. Метод `P2PLibParse` выполняет несложный анализ текста:

1. Разбивает текст на три части, помещая их в массив `msgParts`:
   a. Элемент 0: `"StreamVideo"`
   b. Элемент 1: `" said: "`
   c. Элемент 2: `"C:\Users\Public\Bear.wmv"`
2. Объединяет первые два элемента массива (с индексами 0 и 1), вставляя между ними пробел, и удаляет получившуюся строку из исходного текста:
   a. Исходное сообщение: `"StreamVideo said: C:\Users\Public\Bear.wmv"`
   b. Первые две части: `"StreamVideo said: "`
   c. После выбрасывания результата, получившегося на шаге b, остается только путь: `"C:\Users\Public\Bear.wmv"`
3. Создает новый объект `ChatCommand`, преобразуя элемент `msgParts[0]`, `"StreamVideo"`, в элемент перечисления и передавая в качестве сообщения результат, получившийся на шаге 2.

Пример 5.35. Отбрасывание лишнего текста, добавленного в сообщение (версия на C#)

```
public static ChatMessage P2PLibParse(string text)
{
  //пример:"StreamVideo said: C:\Users\Public\Bear.wmv"
  //разбиваем строку по пробелам
  var msgParts = text.Split(' ');

  //объединяем первые две части текста
  string firstPart = msgParts[0] + " " + msgParts[1] + " ";

  //удаляем получившуюся строку из исходной
  string message = text.Remove(0, firstPart.Length);

  //создаем и возвращаем новый объект сообщения
  ChatMessage msg = new ChatMessage();

  //создаем элемент перечисления ChatCommand
  msg.CommandType = (ChatCommand)Enum.Parse(typeof(ChatCommand), msgParts[0]);
  msg.Message = message;
  return msg;
}
```

Пример 5.36. Отбрасывание лишнего текста, добавленного в сообщение (версия на VB)

```
Public Shared Function P2PLibParse(ByVal text As String) As ChatMessage
  'пример:"StreamVideo said: C:\Users\Public\Bear.wmv"
```

```
    'разбиваем строку по пробелам
    Dim msgParts = text.Split(" "c)

    'объединяем первые две части текста
    Dim firstPart As String = msgParts(0) & " " & msgParts(1) & " "

    'удаляем получившуюся строку из исходной
    Dim message As String = text.Remove(0, firstPart.Length)

    'создаем и возвращаем новый объект сообщения
    Dim msg As New ChatMessage()

    'создаем элемент перечисления ChatCommand
    msg.CommandType = CType(System.Enum.Parse(GetType(ChatCommand),
      msgParts(0)), _ ChatCommand)
    msg.Message = message
    Return msg
  End Function
```

Принципы работы чат-сообщений

Теперь вы знаете, как отправить чат-сообщение по сети и как преобразовать полученную строку в объект типа `ChatMessage`. Далее мы рассмотрим, какие действия должны предпринять клиент и сервер, получив чат-команды в составе события `ChatChanged` (рис. 5.5).

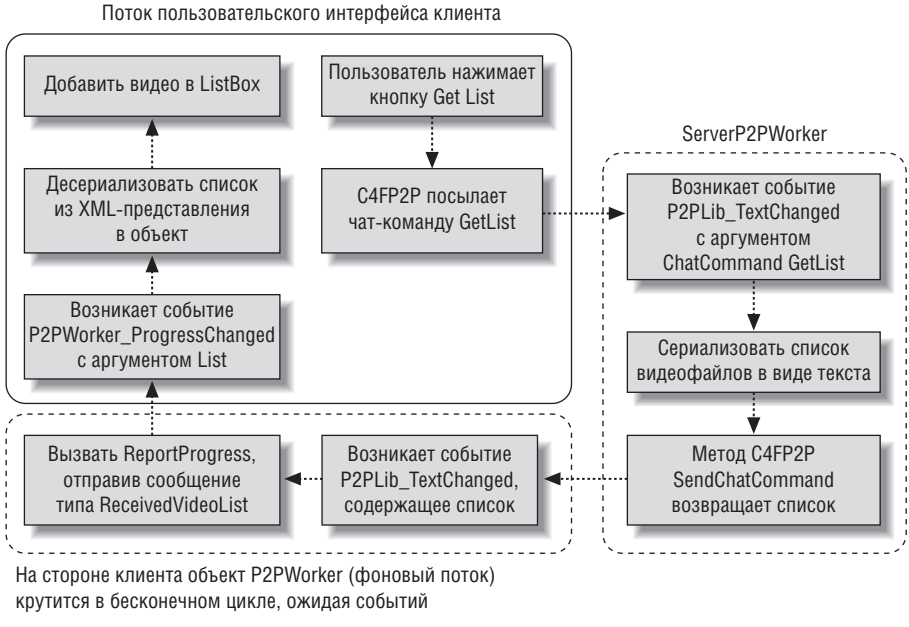

***Рис. 5.5.** Как запрос на получение списка видеофайлов отправляется и принимается по P2P-сети и передается между фоновым потоком и потоком пользовательского интерфейса*

1. В предположении, что PeerCast работает в режиме клиента и соединение с P2P-сетью уже установлено, обмен чат-сообщениями начинается, когда пользователь нажимает кнопку Get List. При этом вызывается метод SendChatMessage менеджера сети, которому передается объект ChatCommand типа GetList (см. примеры 5.88 и 5.89).
2. Метод SendChatMessage отправляет команду GetList по сети, обращаясь к методу SendTextMessage из библиотеки C4FP2P (см. примеры 5.31 и 5.32).
3. Сообщение передается по сети, и в предположении, что на другом ПК PeerCast запущена в режиме сервера, в фоновом потоке возбуждается событие TextChanged, в обработчике которого мы преобразуем полученный текст в объект ChatCommand типа GetList, пользуясь методом P2PLibParse (см. примеры 5.35 и 5.36).
4. Сервер проверяет тип сообщения и начинает обрабатывать запрос GetList. Сначала он преобразует список видеофайлов в текстовый XML-документ (см. примеры 5.45 и 5.46), а затем отправляет этот документ, вызывая метод SendChatMessage менеджера сети и передавая ему команду ReturnList в качестве параметра (см. примеры 5.39 и 5.40).
5. Клиент получает ответ сервера в событии TextChanged и преобразует его в объект ChatMessage типа ReturnList; содержимым сообщения является XML-документ со списком видеофайлов (см. примеры 5.37 и 5.38).
6. Так как список видеофайлов необходимо показать в пользовательском интерфейсе PeerCast, нам придется передать эти данные из фонового потока в поток пользовательского интерфейса с помощью метода ReportProgress. Для этого мы создадим объект ThreadMessage типа ReceivedVideoList и включим в него XML-документ со списком видео (см. примеры 5.39 и 5.40).
7. В потоке пользовательского интерфейса возникает событие ProgressChanged, и мы определяем тип переданного в объекте ThreadMessage сообщения (см. примеры 5.74 и 5.75).
8. Список видео десериализуется, то есть преобразуется из XML-документа в объект класса List, содержащий список строк (см. примеры 5.80 и 5.81).
9. Эти строки добавляются в список видео, отображаемый в элементе управления ListBox, который называется MediaList (см. примеры 5.78 и 5.79).

Получение чат-сообщений в обработчике события ChatChanged

В момент поступления чат-сообщения из сети возбуждается событие P2PLib_ChatChanged с аргументом типа ChatChangeEventArgs. В свойстве e.Message объекта-аргумента и хранится наше сообщение (см. приме-

ры 5.37 и 5.38). Когда это происходит, мы первым делом вызываем метод ReportProgress, чтобы поток пользовательского интерфейса мог отобразить только что полученную команду.

Далее мы смотрим, начинается ли полученное чат-сообщение со строки machine, поскольку нам известно, что это сообщение объект P2PLib автоматически посылает, когда начинает и заканчивает трансляцию видео. Мы не хотим предпринимать никаких действий при получении такого сообщения, поэтому просто выходим из метода ChatChanged.

Если в начале текста сообщения нет строки machine, то это сериализованный объект ChatMessage и мы можем воспользоваться рассмотренным выше методом P2PLibParse, чтобы преобразовать строку в объект ChatMessage. После преобразования можно приступать к обработке сообщений от клиента или сервера. Если PeerCast работает в режиме сервера, то мы вызываем метод ProcessServerMessage. В противном случае проверяем тип полученной команды. Если это команда ReturnList, то мы переправляем сообщение (список видео в формате XML) потоку пользовательского интерфейса для десериализации.

Пример 5.37. Прием команд методом ChatChanged (версия на C#)

```
protected void P2PLib_ChatChanged(object sender, ChatChangedEventArgs e)
{
  //отправляем текст в окно чата
  P2pWorker.ReportProgress(100,
   ThreadMessage.Create(UiMessage.Log, e.Message));

  if (e.Message.StartsWith("machine"))
  {
    return; //выходим
  }

  ChatMessage msg = ChatMessage.P2PLibParse(e.Message);
  if (App.IsServerMode)
  {
    ProcessServerCommand(msg);
  }
  else
  {
    if (msg.CommandType == ChatCommand.ReturnList)
    {
      ChatMessage.P2PLibParse(e.Message);
      P2pWorker.ReportProgress(100,
        ThreadMessage.Create(UiMessage.ReceivedVideoList, msg.Message));
    }
  }
}
```

Пример 5.38. Прием команд методом ChatChanged (версия на VB)

```
Protected Sub P2PLib_ChatChanged(ByVal sender As Object, _
  ByVal e As ChatChangedEventArgs)
```

```
    'отправляем текст в окно чата
    P2pWorker.ReportProgress(100, ThreadMessage.Create(
        UiMessage.Log, e.Message))
    If e.Message.StartsWith("machine") Then
      Return 'выходим
    End If

    Dim msg As ChatMessage = ChatMessage.P2PLibParse(e.Message)
    If App.IsServerMode Then
      ProcessServerCommand(msg)
    Else
      If msg.CommandType = ChatCommand.ReturnList Then
        ChatMessage.P2PLibParse(e.Message)
        P2pWorker.ReportProgress(100, ThreadMessage.Create( _
          UiMessage.ReceivedVideoList, msg.Message))
      End If
    End If
  End Sub
```

Обработка команд на стороне сервера

Если PeerCast работает в режиме сервера, то ее интересуют команды двух типов: `GetList` и `StreamVideo`. Если значение `ChatCommand` соответствует команде `GetList`, то мы вызываем метод `Serializer.SerializeFileList` для сериализации объекта `List`, содержащего список имен видеофайлов, в формат XML. Сериализованный список сервер возвращает клиенту, вызывая метод `SendChatMessage` с параметром `ChatCommand` типа `ReturnList`. Это служит для клиента указанием на то, что получен список видео (см. примеры 5.39 и 5.40).

Вторая команда, интересующая сервер, – это `StreamVideo`. Получив ее, сервер проверяет, есть ли у нас активный объект `videoHttpListener`, и, если передача видео уже идет, то с помощью метода `CancelSendingStream` прерывает трансляцию.

Чтобы начать трансляцию видео клиенту, сервер вызывает метод `SendStreamingData`, передавая ему имя видеофайла. Этот метод начинает читать фрагменты файла, упаковывать их в объекты `StreamPacket` и посылать клиенту по сети.

Пример 5.39. Обработка команд сервером (версия на C#)

```
private void ProcessServerCommand(ChatMessage msg)
{
  switch (msg.CommandType)
  {
    case ChatCommand.GetList:
      string list = Serializer.SerializeFileList();
this.SendChatMessage(ChatMessage.Create(ChatCommand.ReturnList, list));
      break;
    case ChatCommand.StreamVideo:
      if (videoHttpListener != null)
```

```
        {
          //Прерываем текущий поток, если трансляция уже идет
          if (p2p.StreamedVideo.StreamedState ==
                  StreamedStateType.Communicating)
          {
            p2p.StreamedVideo.CancelSendingStream();
          }
          //начинаем трансляцию файла клиенту
          videoHttpListener.SendStreamingData(msg.Message);
        }
        break;
    }
  }
```

Пример 5.40. Обработка команд сервером (версия на VB)

```
Private Sub ProcessServerCommand(ByVal msg As ChatMessage)
  Select Case msg.CommandType
    Case ChatCommand.GetList
      Dim list As String = Serializer.SerializeFileList()
      Me.SendChatMessage(ChatMessage.Create(ChatCommand.ReturnList, list))
    Case ChatCommand.StreamVideo
      If videoHttpListener IsNot Nothing Then
        'Прерываем текущий поток, если трансляция уже идет
        If p2p.StreamedVideo.StreamedState =
               StreamedStateType.Communicating Then
          p2p.StreamedVideo.CancelSendingStream()
        End If
        'начинаем трансляцию файла клиенту
        videoHttpListener.SendStreamingData(msg.Message)
      End If
  End Select
End Sub
```

Чат-команда GetList

Когда PeerCast работает в режиме сервера, она пользуется определяемой на вкладке Settings переменной уровня пользователя `FileDirectory`, чтобы узнать, в каком каталоге хранятся видеофайлы, которые серверу разрешено транслировать (рис. 5.6).

Зная, где искать видеофайлы (значение переменной `FileDirectory`), мы можем вызвать метод `FileUtility.GetVideoList` для поиска WMV-файлов в указанном каталоге (поиск в подкаталогах не ведется), как показано в примерах 5.41 и 5.42. Метод `Directory.GetFiles` возвращает массив строк, который мы преобразуем в универсальный список `List<string>`, вызывая метод расширения `ToList()`.

Пример 5.41. Получение списка видеофайлов (версия на C#)

```
using System.Collections.Generic;
using System.IO;
```

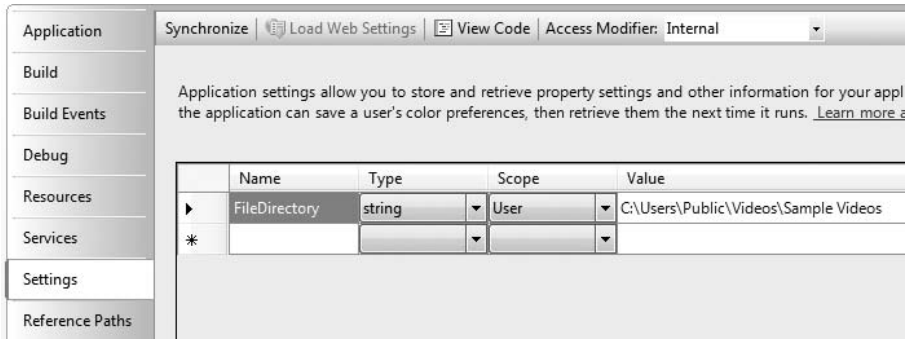

Рис. 5.6. *Задание переменной уровня пользователя, определяющей путь к каталогу с видеофайлами*

```
using System.Linq;

public static class FileUtility
{
  public static List<string> GetVideoList()
  {
    List<string> videos =
      Directory.GetFiles(Properties.Settings.Default.FileDirectory,
      "*.wmv", SearchOption.TopDirectoryOnly).ToList();
    return videos;
  }
}
```

Пример 5.42. Получение списка видеофайлов (версия на VB)

```
Imports Microsoft.VisualBasic
Imports System.Collections.Generic
Imports System.IO
Imports System.Linq

Public Class FileUtility

  Public Shared Function GetVideoList() As List(Of String)
    Dim videos As List(Of String) = Directory.GetFiles( _
      My.Settings.Default.FileDirectory,"*.wmv", _
      SearchOption.TopDirectoryOnly).ToList()
    Return videos
  End Function
End Class
```

Если вызвать метод `FileUtility.GetVideoList()` с принимаемым по умолчанию в ОС Windows Vista значением `FileDirectory` (*C:\Users\Public\Videos\Sample Videos*), то мы получим список `List<string>`, содержащий три элемента, показанные в примере 5.43.

Пример 5.43. Список видео, полученный от GetVideoList

```
[0]: "C:\Users\Public\Videos\Sample Videos\Bear.wmv"
[1]: "C:\Users\Public\Videos\Sample Videos\Butterfly.wmv"
[2]: "C:\Users\Public\Videos\Sample Videos\Lake.wmv"
```

Поскольку нам нужно отправить этот список по сети методом SendChatMessage, который принимает в качестве сообщения только строку, придется сериализовать полученный от GetVideoList список, представив его в формате XML. Для этого мы воспользуемся методом SerializeFileList, который вызывает метод GetVideoList, показанный в примере 5.41, и преобразовывает результат в XML-строку, приведенную в примере 5.44.

Пример 5.44. Сериализованный список видео в формате XML

```
<?xml version="1.0" encoding="utf-8"?>
<ArrayOfString xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <string>C:\Users\Public\Videos\Sample Videos\Bear.wmv</string>
  <string>C:\Users\Public\Videos\Sample Videos\Butterfly.wmv</string>
  <string>C:\Users\Public\Videos\Sample Videos\Lake.wmv</string>
  <string>C:\Users\Public\Videos\Sample Videos\Viva Chihuahuas!.wmv</string>
</ArrayOfString>
```

Для преобразования в формат XML (см. примеры 5.45 и 5.46) мы создаем объект MemoryStream с именем contents, в котором будет храниться сериализованный список файлов; с помощью классов XmlTextWriter и XmlSerializer сериализуем список videoList в формат XML; затем создаем объект UTF8Encoding и в этой кодировке выводим данные из потока MemoryStream в строку.

Понимать все детали работы этих классов необязательно, важно лишь, что мы преобразуем объект-список в XML-строку, которую передаем по P2P-сети.

Пример 5.45. Сериализация списка видеофайлов (версия на C#)

```
using System;
using System.Collections.Generic;
using System.Text;
using System.IO;
using System.Xml.Serialization;
using System.Xml;

public class Serializer
{
  public static string SerializeFileList()
  {
    List<string> videoList = FileUtility.GetVideoList();

    MemoryStream contents = new MemoryStream();
    XmlTextWriter writer = new XmlTextWriter(contents, Encoding.UTF8);

    XmlSerializer xs = new XmlSerializer(typeof(List<string>));
```

```
    xs.Serialize(writer, videoList);

    contents = (MemoryStream)writer.BaseStream;
    UTF8Encoding encoding = new UTF8Encoding();

    return encoding.GetString(contents.ToArray());
  }
...
```

Пример 5.46. Сериализация списка видеофайлов (версия на VB)

```
Imports Microsoft.VisualBasic
Imports System
Imports System.Collections.Generic
Imports System.Text
Imports System.IO
Imports System.Xml.Serialization
Imports System.Xml

Public Class Serializer
  Public Shared Function SerializeFileList() As String
    Dim videoList As List(Of String) = FileUtility.GetVideoList()

    Dim contents As New MemoryStream()
    Dim writer As New XmlTextWriter(contents, Text.Encoding.UTF8)

    Dim xs As New XmlSerializer(GetType(List(Of String)))
    xs.Serialize(writer, videoList)

    contents = CType(writer.BaseStream, MemoryStream)
    Dim encoding As New UTF8Encoding()

    Return encoding.GetString(contents.ToArray())
  End Function
...
```

Применение C4FP2P для трансляции потокового видео

Самое сложное при работе с библиотекой C4FP2P – понять, как передавать видеофайл потоком. Проблема в некоторых ограничениях каркаса Windows Communication Foundation (WCF) и элемента управления `MediaElement`:

Отсутствие встроенной поддержки потоковой трансляции

В Windows Communication Foundation нет встроенной поддержки потоковой трансляции. Следовательно, библиотека C4FP2P должна самостоятельно разбивать видеофайл на последовательно пронумерованные пакеты (объекты типа `StreamPacket`, содержащие массив прочитанных из файла байтов), посылать каждый пакет по сети, а затем собирать из пакетов видеофайл на стороне клиента.

Источником потока должен быть URI

Библиотека P2P проектировалась для поддержки потоковой трансляции в элемент управления `MediaElement`. При этом возникает не-

приятная проблема из-за того, что источником видео в элементе `MediaElement` может быть только Uri, например файл или адрес в Интернете, но не поток байтов в памяти. Разработчики C4FP2P попытались было копировать пакеты в поток, обертывающий файл на диске, а элементу `MediaElement` передавать путь к этому файлу, но потерпели неудачу; как только `MediaElement` обнаруживал конец файла, он прекращал трансляцию. Тогда было решено сконструировать специальный локальный адрес на принимающей машине и записывать массив байтов из пакета `StreamPacket` прямо в соответствующий ему URL. По мере поступления новых пакетов они копируются в URL, а элемент `MediaElement` считывает содержимое этого потока. Поскольку речь идет об открытом HTTP-соединении, `MediaElement` не закрывает его по достижении конца файла, а, следовательно, мы можем начинать и останавливать запись данных в URL.

Принцип работы потокового видео

Сейчас мы попытаемся на концептуальном уровне объяснить, что происходит во время трансляции потокового видео (рис. 5.7).

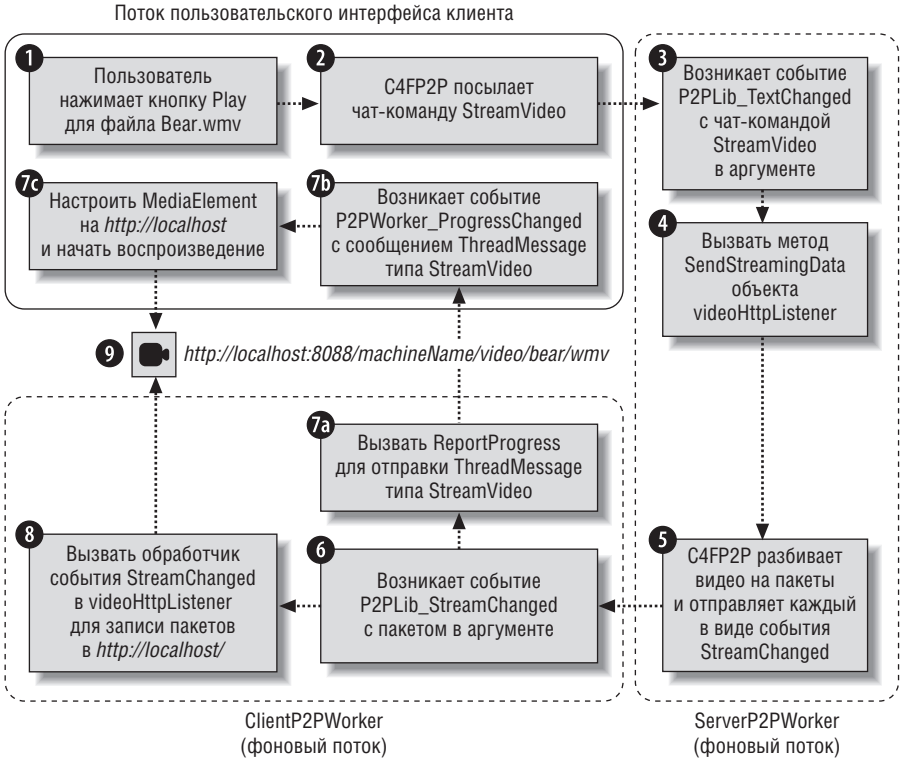

Рис. 5.7. *Как работает потоковое видео*

1. Пользователь, запустивший PeerCast в режиме клиента, выбирает видео, которое желает просмотреть, из списка `MediaList`. Обработчик нажатия на кнопку Play создает объект `ChatMessage` с чат-командой `StreamVideo` и именем выбранного видеофайла в качестве сообщения. Этот объект отправляется в сеть методом `SendChatMessage` класса `NetworkManager` (см. примеры 5.84 и 5.85).
2. Метод `SendChatMessage` отправляет команду `StreamVideo` с указанием имени файла, обращаясь к методу `SendTextMessage` из библиотеки C4FP2P (см. примеры 5.31 и 5.32).
3. Сообщение передается по P2P-сети, и, в предположении, что PeerCast на другом ПК работает в режиме сервера, в фоновом потоке сервера возникает событие `TextChanged`. Мы разбираем полученное текстовое сообщение методом `P2PLibParse`, который преобразует его в объект `ChatCommand` типа `StreamVideo` (см. примеры 5.35 и 5.36).
4. Чтобы начать потоковую трансляцию видеофайла, мы вызываем метод `SendStreamingData` объекта `videoHttpListener`, передавая ему путь к файлу (см. примеры 5.39 и 5.40).
5. В момент создания объекта `videoHttpListener` (см. примеры 5.19 и 5.20) мы зарегистрировали несколько методов из библиотеки C4FP2P в качестве параметров конструктора, например `StartStream` и `SendStream`. Эти методы разбивают видеофайл на сотни мелких пакетов `StreamPacket`. Каждый пакет отсылается узлу-клиенту посредством возбуждения события `StreamChanged`.
6. На стороне клиента при получении каждого пакета `StreamPacket` от сервера возникает событие `StreamChanged`.
7. Получив самый первый пакет `StreamPacket` из события `StreamChanged`, мы должны подготовить пользовательский интерфейс к отображению пакетов на компьютере пользователя.
   a. Чтобы известить поток пользовательского интерфейса о том, что начался прием пакетов видеофайла, мы вызываем метод `ReportProgress` и с его помощью отправляем объект `ThreadMessage` типа `StreamVideo` в поток пользовательского интерфейса. Это сообщение, в частности, содержит URL на машине-клиенте, в который будет направлен видеопоток (см. примеры 5.51 и 5.52).
   b. Возникает событие `ProgressChanged`, и мы определяем тип переданного сообщения `ThreadMessage` (см. примеры 5.74 и 5.75).
   c. Поток пользовательского интерфейса получает объект `ThreadMessage` типа `StreamVideo` и настраивает элемент `MediaElement` на URL *http://localhost*, в который фоновый поток будет направлять объекты `StreamPacket` (см. примеры 5.82 и 5.83).
8. Одновременно с шагом 7, но в фоновом потоке мы переправляем объект `StreamPacket`, полученный на шаге 6, в событие `StreamChanged` объекта `videoHttpListener`. Это событие `StreamChanged` отличается от

события StreamChanged объекта P2PLib (см. раздел «Два разных события StreamChanged»); оно принимает объект StreamPacket и записывает его содержимое (последовательность байтов, представляющих собой кусок видеофайла) по адресу *http://localhost* на машине-клиенте (см. примеры 5.51 и 5.52).

9. Пользователь начинает просмотр видеофайла, транслируемого на его ПК.

Конфигурирование видеопотока

В библиотеке C4FP2P есть два класса, применяемых для потоковой трансляции видеофайлов. Первый называется StreamingHelper. Объект P2PLib раскрывает экземпляр класса StreamingHelper в виде свойства с именем StreamedVideo. Нас будут интересовать два важных метода класса StreamingHelper:

StartStream

Этот метод вызывается, когда вы впервые запрашиваете трансляцию видеофайла. Он инициализирует видеопоток и открывает видеофайл.

SendStream

Этот метод выполняет основную работу: открывает файловый поток, читает его кусками по 12 Кб и отправляет их в сеть в виде объектов StreamPacket.

Знать детали работы этих методов необязательно, нужно только помнить об их существовании, поскольку они передаются в качестве параметров при создании объекта StreamingHttpListener. Сигнатуры методов класса StreamingHttpListener приведены в примерах 5.47 и 5.48.

Пример 5.47. Конструктор класса StreamingHttpListener (версия на C#)

```
public StreamingHttpListener(String uniqueStreamName, _
   WorkerThreadFinished threadFinishedDelegate,LogMessage msg, _
   StartStream startStream, SendStream sendStream)
```

Пример 5.48. Конструктор класса StreamingHttpListener (версия на VB)

```
Public Sub New(ByVal uniqueStreamName As String, _
  ByVal threadFinishedDelegate As WorkerThreadFinished,
  ByVal msg As LogMessage, _
  ByVal startStream As StreamingHttpListener.StartStream, _
  ByVal sendStream As StreamingHttpListener.SendStream)
```

Конструктору класса StreamingListener необходимо передать пять обязательных параметров. Объясним, что они означают.

UniqueStreamName

Это строка для создания уникального локального адреса на машине-клиенте. По этому адресу будет записываться содержимое полу-

чаемых объектов `StreamPacket`. Имя видеопотока задается в следующем формате:

http://localhost:8088/machineName/video/FileName

Например, в случае PeerCast реальный адрес может выглядеть так:

http://localhost:8088/machine633540078269940000/video/bear.wmv

`threadFinishedDelegate`

Этот метод вызывается по завершении трансляции файла. Он полезен для обнаружения ошибки трансляции или для выполнения какого-либо кода по завершении трансляции.

`LogMessage`

Этот метод вызывается, чтобы отправить сообщение, описывающее текущее состояние объекта `StreamingHttpListener`. Ниже приведен список сообщений, передаваемых методу `LogMessage` по ходу потоковой трансляции файла:

- Начиная трансляцию, сервер посылает сообщение «*Sending Media File: filename*»
- Завершая трансляцию, сервер посылает сообщение «*Finished Sending Media*»
- Завершив прием файла, клиент посылает сообщение «*Finished Receiving Media File*»
- Если имела место сетевая ошибка и произошел тайм-аут, клиент посылает сообщение «*Stream Timed out*»

`StartStream`

Этот метод вызывается в начале трансляции файла. Он должен быть присвоен свойству `StreamedVideo.StartStream` объекта `P2PLib`, о котором мы уже говорили выше.

`SendStream`

Этот метод выполняет основную работу по трансляции файла. Он должен быть присвоен свойству `StreamedVideo.SendStream` объекта `P2PLib`, о котором мы уже говорили выше.

Теперь, когда вы понимаете назначение параметров конструктора, обратимся к примерам 5.49 и 5.50, в которых показано, как конструируется объект `videoHttpListener` из примера 5.19.

Пример 5.49. Конструирование объекта videoHttpListener (версия на C#)

```
videoHttpListener = new StreamingHttpListener(userName + "/video/",
  MediaFinished,
  LogMessage,
  p2p.StreamedVideo.StartStream,
  p2p.StreamedVideo.SendStream);
```

Пример 5.50. Конструирование объекта videoHttpListener (версия на VB)

```
videoHttpListener = New StreamingHttpListener(userName & "/video/", _
  AddressOf MediaFinished, AddressOf LogMessage, _
  AddressOf p2p.StreamedVideo.StartStream, _
  AddressOf p2p.StreamedVideo.SendStream)
```

Когда сервер вызывает метод SendStreamingData, чтобы начать потоковую трансляцию видео, библиотека C4FP2P вызывает методы, переданные конструктору объекта videoHttpListener, в том числе LogMessage в момент начала и окончания отправки файла и StartStream и SendStream для разбиения файла на пакеты и отправки объектов StreamPacket клиенту.

О завершении трансляции мы узнаем с помощью события MediaFinished. Чтобы передать данные по сети, метод SendStream посылает объект StreamPacket от сервера клиенту посредством события StreamChanged.

Два разных события StreamChanged

Один из путаных аспектов библиотеки C4FP2P состоит в том, что события StreamChanged определены в двух разных классах. Хотя имя у них одинаковое, по функциональности они не имеют ничего общего!

P2PLib StreamChanged

Это событие StreamChanged служит для отправки объекта StreamPacket от сервера клиенту.

StreamingHttpListener StreamChanged

Обработчик этого события StreamChanged принимает объект StreamPacket и записывает его в URL localhost на клиентской машине.

Неразбериха только увеличивается из-за того, что обработчики обоих событий вызываются прямо один за другим. Сначала мы используем событие P2PLib StreamChanged, чтобы отправить видеопакет от сервера клиенту. А когда клиент получает пакет, возбуждается событие StreamingHttpListener StreamChanged, обработчик которого записывает этот пакет в URL localhost.

Событие P2PLib StreamChanged

Событие P2PLib StreamChanged возникает при получении каждого пакета StreamPacket из сети. Следовательно, мы получим сотни или тысячи таких событий в зависимости от размера видеофайла. Объект StreamPacket передается обработчику в переменной-члене e.StreamedPacket объекта-аргумента StreamChangedEventArgs.

Полученный объект StreamPacket мы передаем обработчику события StreamChanged объекта videoHttpListener. Он в свою очередь записывает пакет в URL *http://localhost*, как показано в примерах 5.51 и 5.52.

Еще одно действие необходимо выполнить при возникновении события в первый раз (то есть при получении пакета StreamPacket, в котором порядковый номер packetNumber равен 0). Необходимо подготовить пере-

менную URL в таком же формате, который используется в классе P2PLib, и отправить этот URL в поток пользовательского интерфейса, чтобы записать его в свойство Source элемента управления MediaElement. Теперь элемент будет читать пакеты из того URL, в который объект P2PLib их записывает.

Пример 5.51. Обработчик события P2PLib StreamChanged (версия на C#)

```
void P2PLib_StreamChanged(object sender, StreamedChangedEventArgs e)
{
  //принять пакет от сервера и с помощью  videoHttpListener
  //записать его в localhost
  videoHttpListener.StreamedChanged(e);

  //настроить клиента на чтение из этого потока
  if (e.StreamedPacket.packetNumber == 0)
  {
    string file = StreamingHttpListener.SafeHttpString(
         e.StreamedPacket.fileName);
    string url = String.Format(@"http://localhost:8088/{0}/video/{1}",
         userName, file);
    P2pWorker.ReportProgress(100, ThreadMessage.Create(UiMessage.StreamVideo,
         url));
  }
}
```

Пример 5.52. Обработчик события P2PLib StreamChanged (версия на VB)

```
Private Sub P2PLib_StreamChanged(ByVal sender As Object, _
  ByVal e As StreamedChangedEventArgs)

  'принять пакет от сервера и с помощью  videoHttpListener
  'записать его в localhost
  videoHttpListener.StreamedChanged(e)

  'настроить клиента на чтение из этого потока
  If e.StreamedPacket.packetNumber = 0 Then
    Dim file As String = StreamingHttpListener.SafeHttpString(_
      e.StreamedPacket.fileName)

    Dim url As String = String.Format("http://localhost:8088/{0}/video/{1}", _
      userName, file)

    P2pWorker.ReportProgress(100, _
      ThreadMessage.Create(UiMessage.StreamVideo, url))
  End If
End Sub
```

Уф! Вот теперь мы честно транслируем видео на адрес *http://localhost*, хранящийся в переменной uniqueStreamName!

Протоколирование сообщений во время трансляции видео

Одним из параметров, которые мы передавали конструктору объекта videoHttpListener (см. примеры 5.49 и 5.50), был метод LogMessage. Он

возбуждает два события: одно в начале трансляции файла сервером, а другое – по завершении. Так, при трансляции видео мы увидим следующие два сообщения:

- Sending Media File: *C:\Users\Public\Videos\Sample Videos\Bear.wmv*
- Finished Sending Media

Сами сообщения генерируются глубоко в недрах класса P2PLib, мы не станем их обрабатывать, а просто переправим в поток пользовательского интерфейса, чтобы зритель видел, чем занято приложение (см. примеры 5.53 и 5.54).

Пример 5.53. Протоколирование сообщений о ходе потоковой трансляции (версия на C#)

```
public void LogMessage(string msg)
{
  P2pWorker.ReportProgress(100, ThreadMessage.Create(ClientMessage.Log, msg));
}
```

Пример 5.54. Протоколирование сообщений о ходе потоковой трансляции (версия на VB)

```
Public Sub LogMessage(ByVal msg As String)
  P2pWorker.ReportProgress(100, ThreadMessage.Create(UiMessage.Log, msg))
End Sub
```

Завершение потоковой трансляции

По завершении трансляции возникает событие `MediaFinished`. В его обработчике мы проверяем, были ли какие-нибудь ошибки, и, если да, то выводим диалоговое окно с сообщением об ошибке, после чего разрываем соединение (см. примеры 5.55 и 5.56).

Пример 5.55. Проверка исключений в событии MediaFinished (версия на C#)

```
protected void MediaFinished(object sender, RunWorkerCompletedEventArgs e)
{
  //Разорвать соединение в случае ошибки
  if (e.Error != null)
  {
    MessageBox.Show(String.Format("Ошибка при трансляции видео:{0}", _
      e.Error.Message));
    EndConnection();
  }
}
```

Пример 5.56. Проверка исключений в событии MediaFinished (версия на VB)

```
Protected Sub MediaFinished(ByVal sender As Object, ByVal e As
RunWorkerCompleted
Protected Sub MediaFinished(ByVal sender As Object, _
  ByVal e As RunWorkerCompletedEventArgs)
  'Разорвать соединение в случае ошибки
  If e.Error IsNot Nothing Then
```

```
    MessageBox.Show(String.Format("Ошибка при трансляции видео:{0}", _
        e.Error.Message))
    EndConnection()
  End If
End Sub
```

Разрыв соединения

Последнее, что осталось сделать в классе `NetworkManager`**, — прибраться по завершении работы. Точнее, мы должны сказать потоку пользовательского интерфейса, чтобы он привел себя в состояние, предшествующее соединению с сетью. Кроме того, мы должны закрыть и сбросить в null (или в Nothing в случае VB) объекты** `P2PLib` **и** `videoHttpListener`**, а также остановить фоновый рабочий поток (см. примеры 5.57 и 5.58).**

Пример 5.57. Разрыв соединения (версия на C#)

```
public void EndConnection()
{
  //это надо сделать только один раз
  if (this.CurrentState != SystemState.LoggedOut)
  {
    this.CurrentState = SystemState.LoggedOut;

    //восстановить пользовательский интерфейс после изменения SystemState
    P2pWorker.ReportProgress(0, ThreadMessage.Create(UiMessage.ToggleUi,
      string.Empty));

    //закрыть объекты и сбросить ссылки в null
    if (p2p != null)
    {
      p2p.Close();
      p2p = null;
    }
    if (videoHttpListener != null)
    {
      videoHttpListener.Stop();
      videoHttpListener.Quit();
      videoHttpListener = null;
    }
    if (P2pWorker != null)
    {
      P2pWorker.CancelAsync();
    }
  }
}
```

Пример 5.58. Разрыв соединения (версия на VB)

```
Public Sub EndConnection()
  'это надо сделать только один раз
  If Me.CurrentState <> SystemState.LoggedOut Then
    Me.CurrentState = SystemState.LoggedOut
```

```
        'восстановить пользовательский интерфейс после изменения SystemState
        P2pWorker.ReportProgress(0, ThreadMessage.Create(UiMessage.ToggleUi, _
          String.Empty))

        'закрыть объекты и сбросить ссылки в Nothing
        If p2p IsNot Nothing Then
          p2p.Close()
          p2p = Nothing
        End If

        If videoHttpListener IsNot Nothing Then
          videoHttpListener.Stop()
          videoHttpListener.Quit()
          videoHttpListener = Nothing
        End If

        If P2pWorker IsNot Nothing Then
          P2pWorker.CancelAsync()
        End If
      End If
    End Sub
```

Конструирование пользовательского интерфейса приложения

Итак, мы рассмотрели, как реализуются сетевые коммуникации в классе `NetworkManager`, но пока еще не создали пользовательский интерфейс PeerCast. Этим мы сейчас и займемся.

Выбор режима работы приложения

При запуске PeerCast открывается окно *ChooseMode.xaml* (рис. 5.8) с двумя кнопками: Client и Server, позволяющими выбрать один из двух режимов работы приложения.

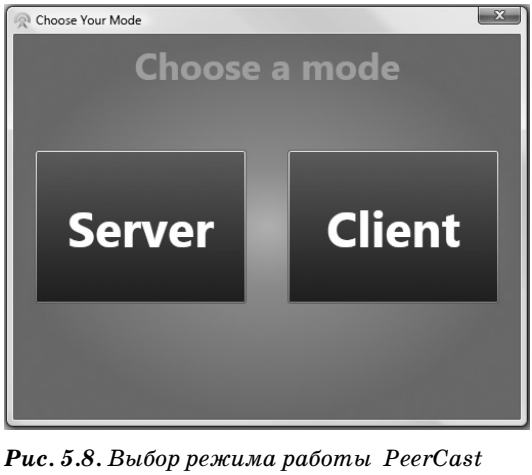

***Рис. 5.8.** Выбор режима работы PeerCast*

При нажатии на ту и другую кнопку вызывается один и тот же обработчик события, `SetMode`, который устанавливает переменную `IsServerMode`, как показано в примерах 5.61 и 5.62. Кроме того, обработчик открывает окно *MainWindow.xaml* и закрывает окно *ChooseMode.xaml*, как показано в примерах 5.59 и 5.60.

Пример 5.59. Обработчик события нажатия на кнопку ChooseMode (версия на C#)

```
private void SetMode(object sender, RoutedEventArgs e)
{
  var button = (Button)sender;
  if (button.Name == "Server")
  {
    App.IsServerMode = true;
  }
  else
  {
    App.IsServerMode = false;
  }

  //открыть главное окно
  MainWindow w = new MainWindow();
  w.Show();

  //закрыть текущее окно
  this.Close();
}
```

Пример 5.60. Обработчик события нажатия на кнопку ChooseMode (версия на VB)

```
Private Sub SetMode(ByVal sender As Object, ByVal e As RoutedEventArgs)
  Dim button = CType(sender, Button)
  If button.Name = "Server" Then
    App.IsServerMode = True
  Else
    App.IsServerMode = False
  End If

  'открыть главное окно
  Dim w As New MainWindow()
  w.Show()

  'закрыть текущее окно
  Me.Close()
End Sub
```

Запоминание режима работы PeerCast

Для хранения режима, в котором работает PeerCast, мы заведем переменную уровня приложения в файле *App.xaml* (*Application.xaml* в случае VB), как показано в примерах 5.61 и 5.62. Режим работы будет

храниться в булевском свойстве `IsServerMode`, которое равно true, если мы работаем в режиме сервера, и false – если в режиме клиента.

Пример 5.61. Определение переменной уровня приложения IsServerMode (версия на C#)

```
using System.Windows;

public partial class App : Application
{
  public static bool IsServerMode;
}
```

Пример 5.62. Определение переменной уровня приложения IsServerMode (версия на VB)

```
Imports Microsoft.VisualBasic
Imports System.Windows
Partial Public Class App
  Inherits Application

  Public Shared IsServerMode As Boolean

End Class
```

Конструирование MainWindow.xaml

Главное окно программы PeerCast состоит из сетки с четырьмя ячейками (рис. 5.9):

Вход в сеть

В левом верхнем углу находится элемент управления для подключения к P2P-сети.

Сообщения

В левом нижнем углу находится область для отображения сообщений, получаемых по сети.

Клиентская область

В правом верхнем углу находятся элементы управления, видимые только в режиме клиента.

Серверная область и медиаплеер

В правом нижнем углу находятся элементы управления, позволяющие сообщить серверу, какие видеофайлы разрешено транслировать клиентам, а также элемент `MediaElement`, используемый в режиме клиента для просмотра видео.

Примечание

PeerCast пользуется файлом *C4fStyle.xaml*, в котором определен внешний вид кнопок, списков ListBox и других элементов управления. Полностью содержимое файла C4fStyle описано в Приложении.

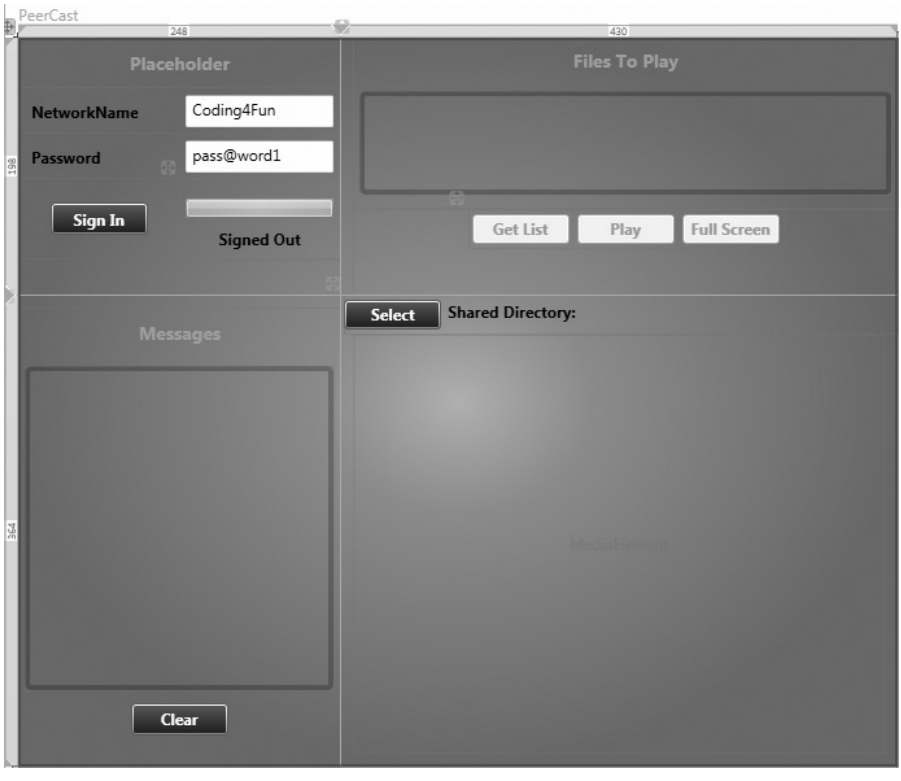

Рис. 5.9. *Главное окно MainWindow с четырьмя ячейками*

Определение сетки с четырьмя ячейками

Для начала определим файл *MainWindow.xaml*, задав основные линейные размеры, а также значок приложения и стиль `SilverGradient` из файла *C4fStyle.xaml*. Затем определим сетку с двумя строками и двумя колонками, внутри которой будут располагаться все элементы управления приложением, и зададим ширину и высоту каждой ячейки (см. пример 5.63).

Пример 5.63. XAML-разметка окна MainWindow

```
<Window x:Class="P2PVideoClient.MainWindow"
  xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
  Title="P2P Video" Height="600" Width="700" MinWidth="500" MinHeight="400"
  Background="{DynamicResource SilverGradient}"
  Icon="/P2PVideoClient;component/icon.png">
  <Grid>
```

```
    <Grid.ColumnDefinitions>
      <ColumnDefinition Width="260*"/>
      <ColumnDefinition Width="450*" />
    </Grid.ColumnDefinitions>
    <Grid.RowDefinitions>
      <RowDefinition Height="198*"></RowDefinition>
      <RowDefinition Height="364*"></RowDefinition>
    </Grid.RowDefinitions>
  </Grid>
</Window>
```

Ячейка входа в сеть

Ячейка входа в сеть, изображенная на рис. 5.10, располагается в левом верхнем углу окна и содержит вложенный элемент `Grid` размером 2×4. Чтобы сообщить подсистеме рендеринга WPF о том, что эта вложенная сетка должна занимать ячейку на пересечении первой строки с первой колонкой (левый верхний угол), мы должны явно присвоить свойствам `Grid.Row` и `Grid.Column` значение 0 (нумерация строк и колонок начинается с нуля).

Первую строку во вложенной сетке занимает метка, занимающая обе колонки; в ней отображается режим работы приложения. Следующие две строки заняты метками и текстовыми полями для ввода имени сети и пароля. В четвертой строке находится кнопка `SignInButton` в первой колонке и вертикально ориентированная панель `StackPanel` с индикатором `progressBar` и расположенной под ним меткой – во второй (пример 5.64).

Пример 5.64. XAML-разметка ячейки входа в сеть

```
<!-- Сетка 2 колонки x 4 строки -->
<Grid Grid.Row="0" Grid.Column="0" Name="Login">
  <Grid.ColumnDefinitions>
    <ColumnDefinition Width="150*"/>
    <ColumnDefinition Width="150*"/>
  </Grid.ColumnDefinitions>
```

***Рис. 5.10.** Ячейка входа в сеть*

```
<Grid.RowDefinitions>
  <RowDefinition Height="Auto"></RowDefinition>
  <RowDefinition Height="Auto"></RowDefinition>
  <RowDefinition Height="Auto"></RowDefinition>
  <RowDefinition Height="Auto"></RowDefinition>
</Grid.RowDefinitions>

<!-- Первая строка -->
<Label Grid.ColumnSpan="2" Grid.Row="0" Margin="5"
  HorizontalAlignment="Center" Name="AppModeLabel"
  Style="{DynamicResource HeaderStyle}">Placeholder</Label>

<!-- Вторая строка -->
<Label Margin="5" Grid.Row="1" Grid.Column="0"
  Name="NetworkNameLabel" Height="25"
  VerticalAlignment="Top" >NetworkName</Label>
<TextBox Grid.Column="1" Grid.Row="1" Margin="5" Name="NetworkName"
  Height="25" VerticalAlignment="Top"></TextBox>

<!-- Третья строка -->
<Label Grid.Column="0" Grid.Row="2" HorizontalAlignment="Left" Margin="5"
  Name="PasswordLabel" Height="25" Width="112">Password</Label>
<TextBox Grid.Column="1" Grid.Row="2" Margin="5" Height="25"
  Name="Password">
</TextBox>

<!-- Четвертая строка-->
<Button Grid.Column="0" Margin="5" Width="75" Height="25" Grid.Row="3"
  Name="SignInButton" Click="SignInButton_Click" >Sign In</Button>
<StackPanel Grid.Column="1" Grid.Row="3">
  <ProgressBar Grid.Column="0" Grid.Row="2" x:Name="progressBar1"
    VerticalAlignment="Top" Margin="5,15,5,0" Height="15"     />
  <Label Grid.Column="1" Grid.Row="2" Margin="3" Name="StatusValue"
    HorizontalContentAlignment="Center">Signed Out</Label>
 </StackPanel>
</Grid>
```

Ячейка сообщений

В этой ячейке (рис. 5.11) отображаются чат-сообщения, полученные из P2P-сети.

Во второй строке первой колонки (левый нижний угол) находится элемент управления `StackPanel` с именем `MessageCell`. Чтобы поместить его в нужную ячейку, мы задаем `Grid.Column` равным 0 (первая колонка), а `Grid.Row` – 1 (вторая строка). Внутри `StackPanel` находится метка `Label`, для которой свойство `Style` задано равным `HeaderStyle` из файла C4fStyle. Все принимаемые сообщения помещаются в список `ListBox` с именем `Messages`. Кроме того, мы включили кнопку `ClearMessages`, которая просто удаляет все сообщения из списка `Messages`. XAML-разметка приведена в примере 5.65.

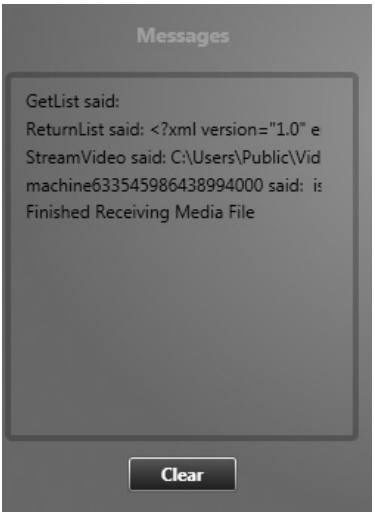

Рис. 5.11. *Ячейка сообщений*

Пример 5.65. XAML-разметка ячейки сообщений

```
<StackPanel Grid.Column="0" Grid.Row="1" Margin="0,10,0,0"
  Name="MessageCell">
  <Label Margin="5" Height="30" Width="120"
    HorizontalContentAlignment="Center"
    Style="{DynamicResource HeaderStyle}">Messages</Label>
  <ListBox Margin="5" Name="Messages" MinHeight="250" MaxHeight="300" />
  <Button Height="25" Margin="5" Name="ClearMessages"
    VerticalAlignment="Bottom" HorizontalAlignment="Center"
    Width="75" Click="ClearMessages_Click">Clear</Button>
</StackPanel>
```

Ячейка с клиентскими элементами управления

В первой строке второй колонки (правый верхний угол) находится набор элементов управления для получения списка видеофайлов и воспроизведения (рис. 5.12).

Как и для ячейки входа в сеть, мы вкладываем внутрь еще одну сетку `Grid` с именем `ClientControls`, состоящую из одной колонки и двух строк. В первой строке находится метка `Label` с заголовком и элемент `ListBox`, который будет содержать список видеофайлов, возвращенный сервером. Во второй строке находится горизонтальная панель `StackPanel` с тремя кнопками:

`GetList`

Отправляет серверу запрос на получение списка имеющихся видеофайлов.

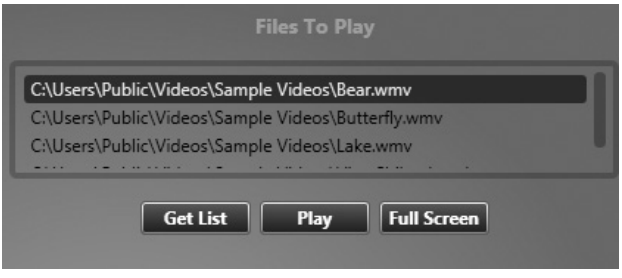

Рис. 5.12. *Ячейка с клиентскими элементами управления*

`Play`

Отправляет серверу запрос на трансляцию видео, выбранного из `MediaList`.

`FullScreen`

Как и `Play`, отправляет запрос на трансляцию выбранного видеофайла, но открывает новое окно *FullScreen.xaml*, в котором и будет воспроизводиться видео.

Эти кнопки сначала деактивированы и становятся активны после получения сообщения `StatusChanged`, означающего, что мы успешно вошли в сеть. Поскольку клиентские элементы управления имеют смысл только при работе в режиме клиента, то при запуске в режиме сервера мы скрываем всю сетку `ClientControls`, присваивая свойству `Visibility` значение `Hidden`. XAML-разметка представлена в примере 5.66.

Пример 5.66. XAML-разметка сетки ClientControls

```
<!-- Вторая колонка -->
<Grid Grid.Column="1" Grid.Row="0" Margin="9.89,0,0,0" Name="ClientControls">
  <Grid.ColumnDefinitions>
    <ColumnDefinition ></ColumnDefinition>
  </Grid.ColumnDefinitions>
  <Grid.RowDefinitions>
    <RowDefinition Height="200*"></RowDefinition>
    <RowDefinition Height="100*"></RowDefinition>
  </Grid.RowDefinitions>

  <!-- Первая строка ClientControls -->
  <StackPanel Grid.Row="0" >
    <Label Height="30" HorizontalAlignment="Center" VerticalAlignment="Top"
      Margin="3" Name="FilesToPlay" Style="{DynamicResource HeaderStyle}">
      Files To Play</Label>
    <ListBox Margin="5" Name="MediaList" MinWidth="300" MinHeight="75"
      MaxHeight="300" IsEnabled="False" RenderTransformOrigin="0.5,0.5"
      Height="80" >
      <ListBox.RenderTransform>
        <TransformGroup>
```

```
            <ScaleTransform ScaleX="1" ScaleY="1"/>
            <SkewTransform AngleX="0" AngleY="0"/>
            <RotateTransform Angle="0"/>
            <TranslateTransform X="0" Y="0"/>
          </TransformGroup>
        </ListBox.RenderTransform>
      </ListBox>
    </StackPanel>

    <!-- Вторая строка ClientControls -->
    <StackPanel Grid.Row="1" Height="30" Orientation="Horizontal"
      VerticalAlignment="Top" HorizontalAlignment="Center">
      <Button Height="23" Margin="3" Name="GetList" Width="75"
        Click="GetList_Click"
        IsEnabled="False" >Get List</Button>
      <Button Height="23" Margin="3" Name="Play" Width="75" Click="Play_Click"
        IsEnabled="False">Play</Button>
       <Button Height="23" Margin="3" Name="FullScreen" Width="75"
         Click="FullScreen_Click" IsEnabled="False">Full Screen</Button>
    </StackPanel>
  </Grid>
```

Ячейка с серверными элементами управления и медиаплеером

Последняя область окна *MainWindow.xaml* – это ячейка на пересечении второй строки со второй колонкой (правый нижний угол). В ней находится элемент для выбора каталога на сервере и медиаплеер (рис. 5.13).

Рис. 5.13. *Ячейка с серверными элементами управления и медиаплеером*

В этой области находится горизонтальная панель StackPanel с именем ServerControls. Она содержит кнопку, открывающую диалоговое окно FolderBrowserDialog, в котором пользователь может выбрать каталог на сервере, где хранятся доступные клиентам файлы. Там же есть метка FilePath, в которой отображается путь к выбранному каталогу. Поскольку каталог задается только при работе в режиме сервера, то при запуске PeerCast в режиме клиента вся панель ServerControls делается невидимой. Ниже панели StackPanel расположен элемент MediaElement с именем MediaPlayer. Именно в нем будет воспроизводиться транслируемое видео после нажатия кнопки Play. XAML-разметка представлена в примере 5.67.

Пример 5.67. XAML-разметка серверных элементов управления и медиаплеера

```
<!-- Серверные элементы управления  -->
<StackPanel Grid.Row="1" Grid.Column="1" Name="ServerControls"
 Margin="3,0,0,0" Height="30" VerticalAlignment="Top"
 Orientation="Horizontal">
  <Button Name="OpenDialog" Click="OpenDialog_Click" Width="75"
    Height="23">Select
  </Button>
  <Label Name="ChooseDirectory">Shared Directory:</Label>
  <Label Name="FilePath" />
</StackPanel>

<!-- Медиаплеер -->
<MediaElement Grid.Row="1" Grid.Column="1" Name="MediaPlayer"
  LoadedBehavior="Manual"
  Margin="10,31,10,10" MinHeight="300" />
```

Добавление программной логики в файл разметки MainWindow.xaml

Определив внешний вид приложения, приступим к заданию свойств и событий для разметки, заданной в файле *MainWindow.xaml*.

Свойства MainWindow

У главного окна MainWindow есть два свойства, показанные в примерах 5.68 и 5.69: одно для класса NetworkManager, которое понадобится при отправке и приеме данных с помощью объекта P2PLib, а другое – для окна FullScreen, создаваемого при нажатии кнопки FullScreen.

Пример 5.68. Свойства MainWindow (версия на C#)

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Linq;
using System.Windows;
using C4F.VistaP2P.Common;
```

```
public partial class MainWindow : Window
{
#region Properties
  //net - обертка всех асинхронных операций с P2P
  private NetworkManager net = new NetworkManager();
  FullScreen fullWindow;
#endregion
...
```

Пример 5.69. Свойства MainWindow (версия на VB)

```
Imports Microsoft.VisualBasic
Imports System
Imports System.Collections.Generic
Imports System.ComponentModel
Imports System.Linq
Imports System.Windows
Imports C4F.VistaP2P.Common

Partial Public Class MainWindow
  Inherits Window

  #Region "Properties"
  'net - обертка всех асинхронных операций с P2P
  Private net As New NetworkManager()
  Private fullWindow As FullScreen
#End Region
```

Конструктор MainWindow

Конструктор `MainWindow` вызывает автоматически генерируемый метод `InitializeComponent()`, а также метод `FirstLoad`, который инициализирует форму `MainWindow`. В методе `FirstLoad` мы проверяем, в каком режиме запущена программа, устанавливаем текст в метке `AppModeLabel` и скрываем элементы управления, относящиеся к другому режиму (при работе в режиме клиента невидимыми делаются элементы, относящиеся к режиму сервера). Если мы работаем в режиме сервера, то в метку `FilePath` записывается значение пользовательской настройки `FileDirectory`, создание которой показано на рис. 5.6. Кроме того, мы подписываемся на события работающего в фоновом потоке объекта `P2pWorker`: `ProgressChanged` и `RunWorkerCompleted`. Это существенно, поскольку событие `ProgressChanged` объекта `P2pWorker` необходимо для отправки обновлений из фонового потока в поток пользовательского интерфейса. Конструктор и метод `FirstLoad` показаны в примерах 5.70 и 5.71.

Пример 5.70. Конструктор MainWindow (версия на C#)

```
public MainWindow()
{
  InitializeComponent();
  FirstLoad();
}
...
```

```
private void FirstLoad()
{
  //Установить свойство Text
  if (App.IsServerMode)
  {
    AppModeLabel.Content = "Server Mode";
    FilePath.Content = Properties.Settings.Default.FileDirectory;
    ClientControls.Visibility = Visibility.Hidden;
  }
  else
  {
    AppModeLabel.Content = "Client Mode";
    ServerControls.Visibility = Visibility.Hidden;
  }

  //Настроить обработчики событий, поскольку все сообщения
  //от NetworkManager проходят через них
  net.P2pWorker.ProgressChanged += new
    ProgressChangedEventHandler(P2PWorker_ProgressChanged);
  net.P2pWorker.RunWorkerCompleted += new System.ComponentModel.
    RunWorkerCompletedEventHandler(P2PBackgroundWorkerCompleted);
}
```

Пример 5.71. Конструктор MainWindow (версия на VB)

```
Public Sub New()
  InitializeComponent()
  FirstLoad()
End Sub

Private Sub FirstLoad()
  'Установить свойство Text
  If App.IsServerMode Then
    AppModeLabel.Content = "Server Mode"
    FilePath.Content = My.Settings.Default.FileDirectory
    ClientControls.Visibility = Visibility.Hidden
  Else
    AppModeLabel.Content = "Client Mode"
    ServerControls.Visibility = Visibility.Hidden
  End If

  'Настроить обработчики событий, поскольку все сообщения
  'от NetworkManager проходят через них
  AddHandler net.P2pWorker.ProgressChanged, _
     AddressOf P2PWorker_ProgressChanged
  AddHandler net.P2pWorker.RunWorkerCompleted, _
     AddressOf P2PBackgroundWorkerCompleted
End Sub
```

Вход в сеть

Чтобы войти в P2P-сеть, мы нажимаем кнопку `SignIn`. Она работает как переключатель в том смысле, что результат зависит от того, установлено ли соединение на этот момент. Если нет, то обработчик прове-

ряет корректность поля NetworkName, а затем устанавливает соединение, передавая имя сети и пароль, как показано в примерах 5.72 и 5.73. Если же вы уже находитесь в сети, то кнопка SignIn разрывает соединение.

Пример 5.72. Вход в P2P-сеть (версия на C#)

```
//Кнопка Sign In устанавливает соединение
private void SignInButton_Click(object sender, RoutedEventArgs e)
{
  if (net.CurrentState == SystemState.LoggedOut)
  {
    if (ValidateFields())
    {
      net.StartConnection(NetworkName.Text, Password.Text);
    }
  }
  else
  {
    net.EndConnection();
  }
}

private bool ValidateFields()
{
  if (string.IsNullOrEmpty(NetworkName.Text) &&
    NetworkName.Text.Trim().Contains(' '))
  {
    MessageBox.Show("Имя сети должно быть непусто и " +
       "состоять только из алфавитно-цифровых символов");
    return false;
  }
  return true;
}
```

Пример 5.73. Вход в P2P-сеть (версия на VB)

```
'Кнопка Sign In устанавливает соединение
Private Sub SignInButton_Click(ByVal sender As Object, ByVal e As
RoutedEventArgs)
  If net.CurrentState = SystemState.LoggedOut Then
    If ValidateFields() Then
      net.StartConnection(NetworkName.Text, Password.Text)
    End If
    Else
      net.EndConnection()
    End If
End Sub

Private Function ValidateFields() As Boolean
  If String.IsNullOrEmpty(NetworkName.Text) AndAlso _
    NetworkName.Text.Trim().Contains(" "c) Then
    MessageBox.Show("Имя сети должно быть непусто и  состоять только из  " & _
      "алфавитно-цифровых символов")
```

```
    Return False
  End If
  Return True
End Function
```

Получение обновлений пользовательского интерфейса в событии ProgressChanged

Мы уже объясняли выше, что PeerCast выполняет все сетевые коммуникации, реализованные в классе `NetworkManager`, в фоновом потоке. Поскольку фоновый поток не вправе напрямую обновлять элементы пользовательского интерфейса, то мы применяем событие `ProgressChanged` как потокобезопасный способ отправить информацию об обновлениях из него в поток пользовательского интерфейса (примеры 5.74 и 5.75). Для этого класс `NetworkManager` отправляет в составе события объект `ThreadMessage` (см. примеры 5.3 и 5.4), включив в него текущее значение индикатора хода процесса. Получив объект `ThreadMessage`, мы проверяем тип сообщения `MessageType` и либо обновляем интерфейс самостоятельно, либо переправляем сообщение другим обработчикам.

Пример 5.74. Отправка сообщений из фонового потока (версия на C#)

```
void P2PWorker_ProgressChanged(object sender, ProgressChangedEventArgs e)
{
  //обновить индикатор хода процесса
  progressBar1.Value = e.ProgressPercentage;

  //прочитать сообщение от NetworkManager
  if (e.UserState != null)
  {
    ThreadMessage message = (ThreadMessage)(e.UserState);
    switch (message.MessageType)
    {
      case UiMessage.UpdateStatus:
        StatusValue.Content = message.Message;
        break;
      case UiMessage.Log:
        Messages.Items.Add(message.Message);
        break;
      case UiMessage.StreamVideo:
        PlayVideo(message.Message);
        break;
      case UiMessage.ReceivedVideoList:
        UpdateVideoList(message.Message);
        break;
      case UiMessage.ToggleUi:
        ToggleUi();
        break;
    }
  }
}
```

Пример 5.75. Отправка сообщений из фонового потока (версия на VB)

```
Private Sub P2PWorker_ProgressChanged(ByVal sender As Object, ByVal e As
Progress-ChangedEventArgs)
  'обновить индикатор хода процесса
  progressBar1.Value = e.ProgressPercentage

  'прочитать сообщение от NetworkManager
  If e.UserState IsNot Nothing Then
    Dim message As ThreadMessage = CType(e.UserState, ThreadMessage)
    Select Case message.MessageType
      Case UiMessage.UpdateStatus
        StatusValue.Content = message.Message
      Case UiMessage.Log
        Messages.Items.Add(message.Message)
      Case UiMessage.StreamVideo
        PlayVideo(message.Message)
      Case UiMessage.ReceivedVideoList
        UpdateVideoList(message.Message)
      Case UiMessage.ToggleUi
        ToggleUi()
    End Select
  End If
End Sub
```

Переключение пользовательского интерфейса

Метод `ToggleUi` вызывается, когда обработчик события `ProgressChanged` получает объект `ThreadMessage` с сообщением `ToggleUi`. В этом случае мы активируем или деактивируем элементы управления в главном окне в зависимости от того, находимся сейчас в сети или нет (примеры 5.76 и 5.77).

Пример 5.76. Переключение пользовательского интерфейса в зависимости от текущего значения SystemState (версия на C#)

```
private void ToggleUi()
{
  if (net.CurrentState == SystemState.LoggedIn ||
    net.CurrentState == SystemState.LoggingIn)
  {
    SignInButton.Content = "Sign Out";
    StatusValue.Content = "Signing In...";
    OpenDialog.IsEnabled = false;
    GetList.IsEnabled = true;
    Play.IsEnabled = true;
    FullScreen.IsEnabled = true;
    MediaList.IsEnabled = true;
  }
  else
  {
    SignInButton.Content = "Sign In";
    StatusValue.Content = "Disconnected";
```

```
    GetList.IsEnabled = false;
    Play.IsEnabled = false;
    FullScreen.IsEnabled = false;
    MediaList.IsEnabled = false;
  }
}
```

Пример 5.77. Переключение пользовательского интерфейса в зависимости от текущего значения SystemState (версия на VB)

```
Private Sub ToggleUi()
  If net.CurrentState = SystemState.LoggedIn OrElse _
    net.CurrentState = SystemState.LoggingIn
  Then
    SignInButton.Content = "Sign Out"
    StatusValue.Content = "Signing In..."
    GetList.IsEnabled = True
    Play.IsEnabled = True
    FullScreen.IsEnabled = True
    MediaList.IsEnabled = True
  Else
    SignInButton.Content = "Sign In"
    StatusValue.Content = "Disconnected"
    GetList.IsEnabled = False
    Play.IsEnabled = False
    FullScreen.IsEnabled = False
    MediaList.IsEnabled = False
  End If
End Sub
```

Получение списка видеофайлов

Метод `UpdateVideoList`, показанный в примерах 5.78 и 5.79, вызывается при получении объекта `ThreadMessage` с сообщением `ReceivedVideoList`. Он десериализует список видеофайлов, посланный в формате XML (см. примеры 5.45 и 5.46), представляя его в виде объекта типа `List<string>`, а затем обходит весь список, добавляя встретившиеся строки в элемент управления `MediaList`.

Пример 5.78. Обновление списка MediaList (версия на C#)

```
private void UpdateVideoList(string serializedList)
{
  List<string> videos = Serializer.DeserializeFileList(serializedList);
  //очистить список
  MediaList.Items.Clear();

  //добавить элементы по одному
  foreach (string s in videos)
  {
    MediaList.Items.Add(s);
  }
}
```

Пример 5.79. Обновление списка MediaList (версия на VB)

```
Private Sub UpdateVideoList(ByVal serializedList As String)
  Dim videos As List(Of String) =
     Serializer.DeserializeFileList(serializedList)
  'очистить список
  MediaList.Items.Clear()

  'добавить элементы по одному
  For Each s As String In videos
    MediaList.Items.Add(s)
  Next s
End Sub
```

Десериализация списка файлов

В классе `Serializer` определен также метод `DeserializeFileList`, противоположный `SerializeFileList` (см. пример 5.45) в том смысле, что он преобразует XML-документ в объект, как показано в примерах 5.80 и 5.81. Этот код конвертирует XML-документ в массив байтов и считывает их в потоковый объект `MemoryStream`. Теперь мы можем воспользоваться классами `XmlTextWriter` и `XmlSerializer`, чтобы десериализовать XML в список строк. Понимать все детали необязательно; важно понимать, что мы берем строку в формате XML и строим из нее объект `List`.

Пример 5.80. Преобразование XML-строки в объект List<string> (версия на C#)

```
public static List<string> DeserializeFileList(string serializedList)
{
  //десериализовать список файлов
  UTF8Encoding encoding = new UTF8Encoding();
  Byte[] byteArray = encoding.GetBytes(serializedList);
  MemoryStream contents = new MemoryStream(byteArray);
  XmlTextWriter writer = new XmlTextWriter(contents, Encoding.UTF8);
  XmlSerializer xs = new XmlSerializer(typeof(List<string>));

  return (List<string>) xs.Deserialize(contents);
}
```

Пример 5.81. Преобразование XML-строки в объект List<string> (версия на VB)

```
Public Shared Function DeserializeFileList(ByVal serializedList As String) As _
  List(Of String)
  ' десериализовать список файлов
  Dim encoding As New UTF8Encoding()
  Dim byteArray() As Byte = encoding.GetBytes(serializedList)
  Dim contents As New MemoryStream(byteArray)
  Dim writer As New XmlTextWriter(contents, Text.Encoding.UTF8)
  Dim xs As New XmlSerializer(GetType(List(Of String)))

  Return CType(xs.Deserialize(contents), List(Of String))
End Function
```

Воспроизведение видео

Метод `PlayVideo` вызывается при получении объекта `ThreadMessage` с сообщением `StreamVideo`. Он выясняет, где надо воспроизводить видео: в окне `fullWindow`, занимающем весь экран, или в медиаплеере внутри окна `MainWindow`, а затем просто устанавливает URL видеопотока, как показано в примерах 5.82 и 5.83.

Пример 5.82. Настройка элементов управления MediaPlayer (версия на C#)

```
public void PlayVideo(string url)
{
  if (fullWindow == null)
  {
    MediaPlayer.Source = new Uri(url);
    MediaPlayer.Play();
  }
  else
  {
    fullWindow.SetMediaPlayer(new Uri(url));
    fullWindow.Show();
  }
}
```

Пример 5.83. Настройка элементов управления MediaPlayer (версия на VB)

```
Public Sub PlayVideo(ByVal url As String)
  If fullWindow Is Nothing Then
    MediaPlayer.Source = New Uri(url)
    MediaPlayer.Play()
  Else
    fullWindow.SetMediaPlayer(New Uri(url))
    fullWindow.Show()
  End If
End Sub
```

Отправка сообщения о начале трансляции

Когда пользователь нажимает кнопку `Play`, мы с помощью метода `SendChatMessage` класса `NetworkManager` передаем указание начать потоковую трансляцию видеофайла, выбранного из списка `MediaList` (примеры 5.84 и 5.85).

Пример 5.84. Отправка чат-сообщения с указанием начать трансляцию выбранного видео (версия на C#)

```
private void Play_Click(object sender, RoutedEventArgs e)
{
  // отправить запрос на трансляцию видео
  net.SendChatMessage(ChatMessage.Create(ChatCommand.StreamVideo,
  (string)MediaList.SelectedItem));
}
```

Пример 5.85. Отправка чат-сообщения с указанием начать трансляцию выбранного видео (версия на VB)

```
Private Sub Play_Click(ByVal sender As Object, ByVal e As RoutedEventArgs)
  ' отправить запрос на трансляцию видео
  net.SendChatMessage(ChatMessage.Create(ChatCommand.StreamVideo, CStr( _
    MediaList.SelectedItem)))
End Sub
```

Отправка сообщения о начале полноэкранной трансляции

Когда пользователь нажимает кнопку `Full Screen`, мы проверяем, создано ли на этот момент окно `fullWindow`. Если нет, мы создаем его и устанавливаем обработчик события `Closing`, который присвоит переменной `fullWindow` значение null, как показано в примерах 5.86 и 5.87. Это необходимо, чтобы избежать исключения из-за нулевой ссылки, которое может возникнуть, если пользователь закроет окно `fullWindow`, а потом еще раз нажмет кнопку `Full Screen`. Как и в описанном выше случае с кнопкой `Play`, мы отправляем имя выбранного видеофайла серверу, чтобы он начал трансляцию.

Пример 5.86. Воспроизведение видео в отдельном полноэкранном окне (версия на C#)

```
private void FullScreen_Click(object sender, RoutedEventArgs e)
{
  //открыть новое окно на весь экран
  if (fullWindow == null)
  {
    fullWindow = new FullScreen();
    fullWindow.Closing += new CancelEventHandler(fullWindow_Closing);
  }
  net.SendChatMessage(ChatMessage.Create(ChatCommand.StreamVideo,
    (string)MediaList.SelectedItem));
}

void fullWindow_Closing(object sender, CancelEventArgs e)
{
  //присвоить null во избежание попыток воспроизвести видео
  //в закрытом окне, что привело бы к возникновению исключения
  fullWindow = null;
}
```

Пример 5.87. Воспроизведение видео в отдельном полноэкранном окне (версия на VB)

```
Private Sub FullScreen_Click(ByVal sender As Object, ByVal e As
RoutedEventArgs)
  'открыть новое окно на весь экран
  If fullWindow Is Nothing Then
    fullWindow = New FullScreen()
    AddHandler fullWindow.Closing, AddressOf fullWindow_Closing
  End If
```

```
    net.SendChatMessage(ChatMessage.Create(ChatCommand.StreamVideo, _
      CStr(MediaList.SelectedItem)))
End Sub

Private Sub fullWindow_Closing(ByVal sender As Object, ByVal e As
CancelEventArgs)
    'присвоить null во избежание попыток воспроизвести видео
    'в закрытом окне, что привело бы к возникновению исключения
    fullWindow = Nothing
End Sub
```

Получение списка видеофайлов от сервера

Когда пользователь нажимает кнопку `GetList`, мы отсылаем серверу чат-команду с просьбой вернуть список видеофайлов, как показано в примерах 5.88 и 5.89. Сервер отправляет нам список в формате XML, который мы получим в составе события `ProgressChanged`. Поскольку никаких данных в сообщении передавать не надо, то мы записываем в параметр `message` пустую строку.

Пример 5.88. Отправка чат-сообщения с запросом на получение списка видеофайлов (версия на C#)

```
private void GetList_Click(object sender, RoutedEventArgs e)
{
  //отправить запрос на получение списка видеофайлов
  net.SendChatMessage(ChatMessage.Create(ChatCommand.GetList,
    string.Empty));
}
```

Пример 5.89. Отправка чат-сообщения с запросом на получение списка видеофайлов (версия на VB)

```
Private Sub GetList_Click(ByVal sender As Object, ByVal e As RoutedEventArgs)
  //отправить запрос на получение списка видеофайлов
  net.SendChatMessage(ChatMessage.Create(ChatCommand.GetList, String.Empty))
End Sub
```

Выбор каталога с видеофайлами

Если PeerCast запущена в режиме сервера, то вы можете изменить принимаемый по умолчанию каталог для хранения видеофайлов, воспользовавшись классом `System.Windows.Forms FolderBrowserDialog`, как показано в примерах 5.90 и 5.91. Когда пользователь выберет новый каталог и нажмет `OK`, путь к выбранному каталогу будет записан в переменную `FileDirectory`, которую мы определили, как показано на рис. 5.6. Так как `FileDirectory` объявлена пользовательской настройкой, то Visual Studio сама позаботится о сохранении нового значения, не заставляя нас заниматься ручной сериализацией и десериализацией.

Пример 5.90. Открытие диалогового окна FolderBrowser (версия на C#)

```
private void OpenDialog_Click(object sender, RoutedEventArgs e)
{
  var folder = new System.Windows.Forms.FolderBrowserDialog();
  folder.RootFolder = Environment.SpecialFolder.MyComputer;

  System.Windows.Forms.DialogResult r = folder.ShowDialog();
  if (r == System.Windows.Forms.DialogResult.OK)
  {
    Properties.Settings.Default.FileDirectory = folder.SelectedPath;
    Properties.Settings.Default.Save();
    FilePath.Content = folder.SelectedPath;
  }
}
```

Пример 5.91. Открытие диалогового окна FolderBrowser (версия на VB)

```
Private Sub OpenDialog_Click(ByVal sender As Object, ByVal e As
RoutedEventArgs)
  Dim folder = New System.Windows.Forms.FolderBrowserDialog()
  folder.RootFolder = Environment.SpecialFolder.MyComputer

  Dim r As System.Windows.Forms.DialogResult = folder.ShowDialog()
  If r = System.Windows.Forms.DialogResult.OK Then
    My.Settings.Default.FileDirectory = folder.SelectedPath
    My.Settings.Default.Save()
    FilePath.Content = folder.SelectedPath
  End If
End Sub
```

Когда поток P2pWorker завершается

Вернемся к методу `FirstLoad` (пример 5.70). В числе обработчиков, установленных для объекта `P2pWorker`, был, в частности, и обработчик события `BackgroundWorkerCompleted`, которое возникает, когда фоновый поток `NetworkManager` завершает работу (см. примеры 5.92 и 5.93). Напомним (см. примеры 5.23 и 5.24), что в фоновом потоке исполняется бесконечный цикл, следовательно, это событие возбуждается только в случае, когда пользователь закрывает соединение или в фоновом потоке происходит ошибка.

Пример 5.92. Обработчик события Completed объекта P2pWorker (версия на C#)

```
protected void P2pBackgroundWorkerCompleted(object sender,
   RunWorkerCompletedEventArgs e)
{
  //произошла ошибка
  if (e.Error != null)
  {
    MessageBox.Show(String.Format("Произошла ошибка в фоновом потоке: {0}",
       e.Error.Message));
  }
```

```
    else
    {
      net.EndConnection();
    }
  }
```

Пример 5.93. Обработчик события Completed объекта P2pWorker (версия на VB)

```
Protected Sub P2pBackgroundWorkerCompleted(ByVal sender As Object, ByVal e As _
  RunWorkerCompletedEventArgs)
  'произошла ошибка
  If e.Error IsNot Nothing Then
    MessageBox.Show(String.Format("Произошла ошибка в фоновом потоке: {0}",
      e.Error.Message))
  Else
    net.EndConnection()
  End If
End Sub
```

Заключительные замечания

Создание P2P-приложений может оказаться сложным делом, учитывая, сколько всего надо сделать: от многопоточного программирования до освоения новых сетевых протоколов. Но в награду мы получаем потенциальную мощь пиринговых сетей. Преимущества передачи объектов, видеофайлов и прочих активов в Интернете децентрализованным, безопасным и масштабируемым способом трудно переоценить. Вы можете разработать потрясающие приложения, обладая лишь пониманием механизма работы событий и толикой воображения.

6

TwitterVote

Автор	Клинт Руткас (*http://betterthaneveryone.com/*), Дэн Фернандес и Брайан Пик
Сложность	Средняя
Необходимое время	3–6 часов
Стоимость	Бесплатно
Программное обеспечение	Visual Web Developer 2008 Express Edition
Оборудование	Не требуется
Адрес в Интернете	*http://www.c4fbook.com/TwitterVote*

Twitter (*http://www.twitter.com*) – это веб-приложение, позволяющее друзьям, родственникам, коллегам и вообще кому угодно обмениваться короткими сообщениями, используя броузер, SMS и другие клиентские приложения. Многие называют Twitter платформой для микроблоггинга, но это также и вид открытой, массовой коммуникации в формате, близком к Интернет-пейджерам. Сообщение, отправляемое серверу Twitter, называется «твит». В этой главе мы будем рассматривать сообщения об изменении статуса как твиты.

Twitter – потрясающий инструмент, обладающий рядом несомненных достоинств: он бесплатен, сформировал внушающее уважение сообщество и предлагает довольно мощный API.

В этой главе мы разработаем приложение TwitterVote, позволяющее создавать онлайновые опросы, в которых другие пользователи смогут проголосовать через Twitter. Например, можно спросить своих дру-

зей, какую игровую приставку они предпочитают: Wii, PS3 или Xbox360. Участники опроса голосуют, а вы видите итоги голосования.

Краткий обзор

Для работы с Twitter вам понадобится создать бесплатную учетную запись. Для этого зайдите на сайт Twitter по адресу *http://www.twitter.com/*. Выберите легко запоминающиеся имя и пароль, поскольку они вам понадобятся при создании TwitterVote.

Создав учетную запись, войдите в систему и выберите службу. После входа окно будет выглядеть примерно так, как показано на рис. 6.1.

Вы можете изменить свой статус, набрав в текстовом поле сверху сообщение и нажав кнопку update.

В Twitter вы можете отвечать конкретному пользователю в формате *@targetUsername Это ответ*. Такие ответы адресат увидит на вкладке Replies в своем броузере.

Ценность Twitter возрастает, когда вы начинаете искать других людей и подписываетесь на их твиты, а также по мере присоединения к сети

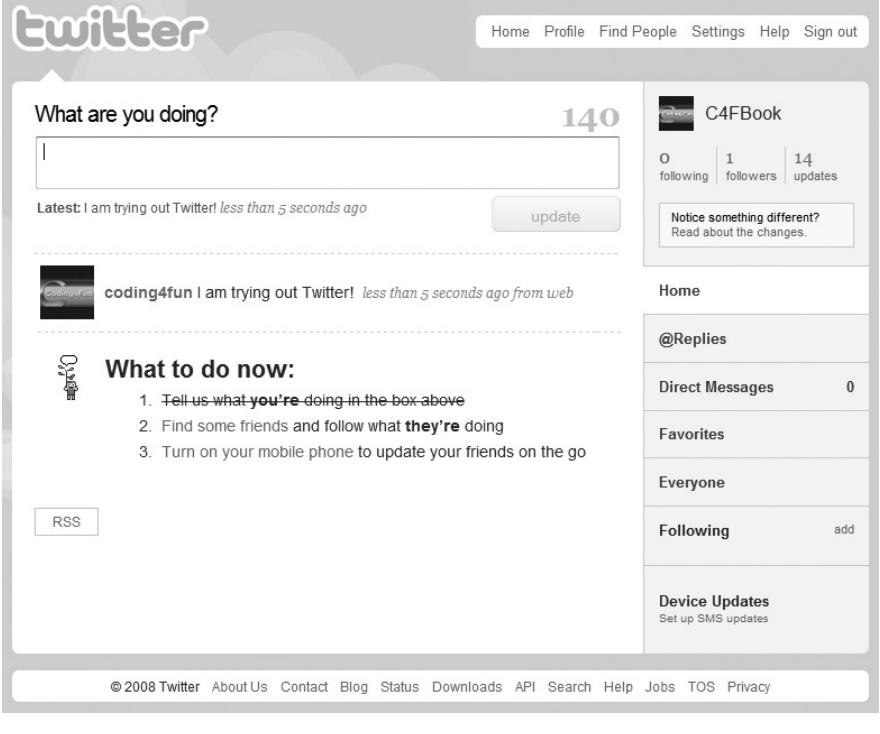

Рис. 6.1. *Пользовательский интерфейс Twitter*

ваших друзей. Вы даже можете подписаться на твиты сайта Coding-4Fun, добавив имя @coding4fun (*http://twitter.com/coding4fun*). Ну а теперь займемся созданием приложения TwitterVote.

Правила и ограничения Twitter

У Twitter есть несколько правил и ограничений. Одно из них заключается в том, что длина сообщения не должна превышать 140 символов. Это ограничение существенно влияет на то, для каких целей можно использовать Twitter. Оно ставит перед вами задачу создания сообщений одновременно кратких и выразительных. Кроме того, с помощью специальных приемов Twitter API ограничивает количество сообщений, отправляемых одним пользователем в час. В настоящее время верхняя граница составляет 100 сообщений в час, но в будущем она может измениться. Имейте это в виду, когда будете отправлять запросы этому или другому приложению Twitter.

Проектирование TwitterVote

Немного познакомившись с самой службой Twitter, давайте поговорим о приложении, которое мы собираемся написать. Это веб-приложение будет искать в вашей собственной учетной записи ключевое слово, показывающее, что опрос начался, а также вид данного опроса (ввод числа или выбор из списка текстовых значений).

Затем приложение просмотрит список ответов, поступивших в вашу учетную запись, и решит, какие из них относятся к голосованию в опросе. Отобранные ответы будут выведены на веб-страницу в виде таблицы.

Twitter: под капотом

Twitter API следует архитектурному стилю REST (Representational State Transfer – передача представляемых состояний). Проще говоря, API раскрывается через веб и возвращает данные в строго определенном формате. Ниже приведены некоторые примеры Twitter API. Отметим, что данные могут быть представлены в формате XML, JSON, RSS или ATOM в зависимости от вызова API. Для запроса данных в другом формате просто подставьте вместо расширения *.xml* одно из расширений *.json*, *.rss* или *.atom*.

- Твиты пользователя: *http://twitter.com/statuses/user_timeline/coding4fun.xml*
- Друзья пользователя: *http://twitter.com/statuses/friends_timeline.xml*
- Ответы пользователя: *http://twitter.com/statuses/replies.xml*
- Счетчик вызовов API: *http://twitter.com/account/rate_limit_status.xml*

Twitter API

Модель Twitter API описана на странице *http://apiwiki.twitter.com/*. В табл. 6.1 перечислены поддерживаемые в настоящее время вызовы API.

Таблица 6.1. Методы Twitter API

Группа методов	Методы
Информация о состоянии	`public_timeline`, `friends_timeline`, `user_timeline`, `show`, `update`, `replies`, `destroy`
Информация о пользователях	`friends`, `followers`, `featured`, `show`
Прямые сообщения	`direct_messages`, `sent`, `new`, `destroy`
Учетная запись	`verify_credentials`, `end_session`, `archive`, `update_location`, `update_delivery_device`
Избранное	`favorites`, `create`, `destroy`
Извещения	`follow`, `leave`
Блокировка	`create`, `destroy`
Вспомогательные	`est`, `downtime_schedule`

В приложении TwitterVote будут использоваться методы из групп *User* и *Status*. Для их вызова следует отправить запрос на сайт Twitter, передав имя метода в качестве первого слова после косой черты в URL. Например, чтобы вызвать метод `user_timeline` для пользователя C4FBook, нужно отправить запрос *http://twitter.com/statuses/user_timeline/c4fbook.xml*. Если ввести этот URL в броузере, то вы получите в ответ XML-документ, показанный в примере 6.1.

Пример 6.1. XML-документ, возвращенный для пользователя C4FBook

```
<?xml version="1.0" encoding="UTF-8"?>
<statuses type="array">
<status>
    <created_at>Sun Aug 10 07:14:47 +0000 2008</created_at>
    <id>883094245</id>
    <text>I am trying out Twitter!</text>
    <source>web</source>
    <truncated>false</truncated>
    <in_reply_to_status_id></in_reply_to_status_id>
    <in_reply_to_user_id></in_reply_to_user_id>
    <favorited></favorited>
    <user>
      <id>15796176</id>
      <name>C4FBook</name>
      <screen_name>C4FBook</screen_name>
      <location></location>
      <description></description>
```

```
      <profile_image_url>http://static.twitter.com/images/
                              /default_profile_normal.png
      </profile_image_url>
      <url></url>
      <protected>false</protected>
      <followers_count>0</followers_count>
    </user>
</status>
</statuses>
```

В числе прочего здесь содержится и последний твит пользователя C4FBook:

```
<text>I am trying out Twitter!</text>
```

Реализация Twitter

Первым делом откройте Visual Web Developer Express 2008 и создайте приложение типа *ASP.NET Web Application* на языке C# или Visual Basic, назвав его *TwitterVote*, как показано на рис. 6.2.

Когда приложение будет создано, щелкните правой кнопкой мыши по проекту TwitterVote, выберите из контекстного меню пункт New Folder и создайте новую папку *App_Code*, как показано на рис. 6.3. Здесь будут находиться все относящиеся к проекту файлы с кодом.

Теперь напишем код для обращения к Twitter и разбора результатов. Добавьте в папку *App_Code* новый класс `Tweet`. Это довольно простой класс со свойствами, в которых будут храниться данные о твите, отправленном пользователем Twitter. Его код представлен в примерах 6.2 и 6.3.

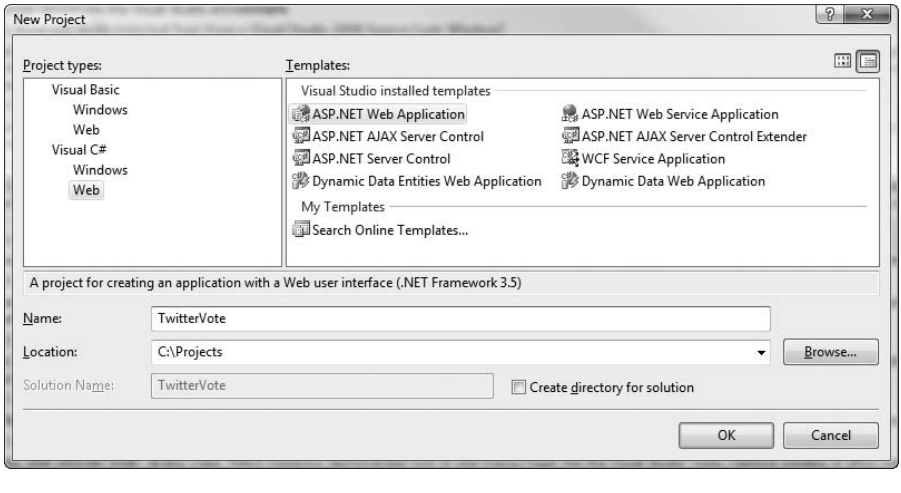

Рис. 6.2. *Создание приложения типа ASP.NET Web Application*

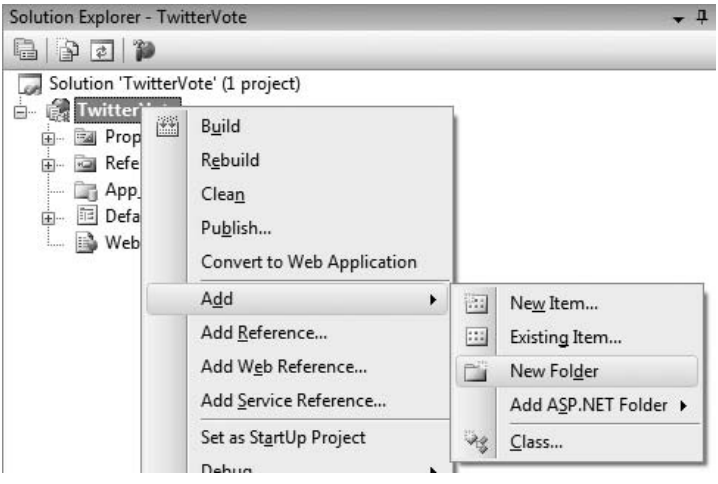

Рис. 6.3. *Создание папки App_Code*

Пример 6.2. Класс Tweet (версия на C#)

```
using System;

public class Tweet
{
  public string Text { get; set; }
  public DateTime? DateCreated { get; set; }
  public string UserName { get; set; }
}
```

Пример 6.3. Класс Tweet (версия на VB)

```
Imports Microsoft.VisualBasic
Imports System

Public Class Tweet
  Private privateText As String
  Public Property Text() As String
    Get
      Return privateText
    End Get
    Set(ByVal value As String)
      privateText = value
    End Set
  End Property

  Private privateDateCreated As Nullable(Of DateTime)
  Public Property DateCreated() As Nullable(Of DateTime)
    Get
      Return privateDateCreated
    End Get
    Set(ByVal value As Nullable(Of DateTime))
```

```
      privateDateCreated = value
    End Set
  End Property

  Private privateUserName As String
  Public Property UserName() As String
    Get
      Return privateUserName
    End Get
    Set(ByVal value As String)
      privateUserName = value
    End Set
  End Property
End Class
```

Теперь можно написать класс, который будет получать данные от Twitter, разбирать их и строить объекты `Tweet`. Создайте в каталоге *App_Code* новый класс `TwitterService`.

Его конструктор принимает имя и пароль того пользователя Twitter, который обращается к службе. Мы уже отмечали, что нам понадобятся методы API `user_timeline` и `replies`, поэтому включим в наш класс соответствующие им методы, которые будут вызываться, чтобы передать данные из методов Twitter, как показано в примерах 6.4 и 6.5.

Пример 6.4. Класс TwitterService (версия на C#)

```
using System;
using System.Collections.Generic;
using System.IO;
using System.Xml;
using System.Linq;
using System.Net;
using System.Xml.Linq;
using System.Security;
using System.Globalization;

public class TwitterService
{
  private string username, password;
  public TwitterService(string user, string pass)
  {
    // сохраняем имя и пароль пользователя twitter
    username = user;
    password = pass;
  }

  public List<Tweet> GetUserTimeline()
  {
    string url =
        string.Format("http://twitter.com/statuses/user_timeline/{0}.xml",
                username);
        return ConvertTwitterXmlToList(new Uri(url));
```

```
    }

    public List<Tweet> GetReplies()
    {
    // ИЗВОРАЧИВАЕМСЯ: Из-за ошибки в Twitter приходится пока
    // поступать так...
    int random = (int)(new Random().NextDouble() * 100000);
    return ConvertTwitterXmlToList(new Uri(
        string.Format("http://twitter.com/statuses/replies.xml?since_id={0}",
                      random)));
    }
}
```

Пример 6.5. Класс TwitterService (версия на VB)

```
Imports Microsoft.VisualBasic
Imports System
Imports System.Collections.Generic
Imports System.IO
Imports System.Xml
Imports System.Linq
Imports System.Net
Imports System.Xml.Linq
Imports System.Security
Imports System.Globalization

Public Class TwitterService
  Private username, password As String

  Public Sub New(ByVal user As String, ByVal pass As String)
    ' сохраняем имя и пароль пользователя twitter
    username = user
    password = pass
  End Sub

  Public Function GetUserTimeline() As List(Of Tweet)
    Dim url As String = _
       String.Format("http://twitter.com/statuses/user_timeline/{0}.xml", _
          username)
    Return ConvertTwitterXmlToList(New Uri(url))
  End Function

  Public Function GetReplies() As List(Of Tweet)
    ' ИЗВОРАЧИВАЕМСЯ: Из-за ошибки в Twitter приходится пока
    ' поступать так...
    Dim random As Integer = CInt(Fix(New Random().NextDouble() * 100000))
    Return ConvertTwitterXmlToList(New Uri( _
       String.Format("http://twitter.com/statuses/replies.xml?since_id={0}", _
                     random)))
  End Function
End Class
```

Оба эти метода просто обращаются к методу `ConvertTwitterXmlToList`**, который мы вскоре напишем, и передают ему объект типа** `Uri`**, задающий путь к методу Twitter API.**

Обратите внимание на кусок встроенного кода в методе `GetReplies`. В Twitter API есть ошибка, из-за которой ответы неправильно кэшируются, что приводит к печальным последствиям в нашем приложении. Чтобы обойти ее, мы включаем в параметр `since_id` случайное значение, заставляя Twitter каждый раз посылать настоящие, а не кэшированные результаты. Когда ошибка будет исправлена, от этого трюка можно будет отказаться.

Метод `ConvertTwitterXmlToList` отправит на указанный Uri запрос, получит в ответ XML-документ, разберет его на объекты `Tweet` и вернет их в виде универсального .NET-объекта `List`. Все это показано в примерах 6.6 и 6.7.

Пример 6.6. Метод ConvertTwitterXmlToList (версия на C#)

```
public List<Tweet> ConvertTwitterXmlToList(Uri TwitterUrl)
{
  Stream s = null;

  // вызываем службу
  WebClient wc = new WebClient();
  wc.Credentials = new NetworkCredential(username, password);

  // получаем данные
  try
  {
    s = wc.OpenRead(TwitterUrl);
  }
  catch (WebException we)
  {
    // если ошибка
    if(we.Status == WebExceptionStatus.ProtocolError)
    {
      // получить код состояния HTTP
      switch((int)((HttpWebResponse)we.Response).StatusCode)
      {
        // ошибка HTTP 400 == превышена частота обращения к twitter
        case 400:
          throw new ApplicationException(
               "Twitter rate limit reached...please try again later.");
        // ошибка HTTP 401 == не пройдена авторизация
        case 401:
          throw new SecurityException("User not authorized");
        default:
          throw;
      }
    }
  }

  XmlTextReader xr = new XmlTextReader(s);
  XDocument rawData = XDocument.Load(xr);

  // поместить данные из каждого элемента status в новый объект Tweet
```

```
 var query = from entry in rawData.Descendants("statuses").
                                   Descendants("status")
             select new Tweet
             {
              UserName = entry.Element("user").Element("screen_name").Value,
              Text = entry.Element("text").Value,
              // Дата в формате HTTP
              DateCreated = DateTime.ParseExact(
                       entry.Element("created_at").Value,
                       "ddd MMM dd HH:mm:ss zzzz yyyy",
                       CultureInfo.GetCultureInfoByIetfLanguageTag("en-us"),
                       DateTimeStyles.AllowWhiteSpaces)
             };

  return query.ToList();
}
```

Пример 6.7. Метод ConvertTwitterXmlToList (версия на VB)

```
Public Function ConvertTwitterXmlToList(ByVal TwitterUrl As Uri)
As List(Of Tweet)
  Dim s As Stream = Nothing

  ' вызываем службу
  Dim wc As New WebClient()
  wc.Credentials = New NetworkCredential(username, password)

  ' получаем данные
  Try
    s = wc.OpenRead(TwitterUrl)
  Catch we As WebException
    ' если ошибка
    If we.Status = WebExceptionStatus.ProtocolError Then
      ' получить код состояния HTTP
     Select Case CInt(Fix((CType(we.Response, HttpWebResponse)).StatusCode))
        ' ошибка HTTP 400 == превышена частота обращения к twitter
        Case 400
          Throw New ApplicationException(_
               "Twitter rate limit reached...please try again later.")
        ' ошибка HTTP 401 == не пройдена авторизация
        Case 401
          Throw New SecurityException("User not authorized")
        Case Else
          Throw
      End Select
    End If
  End Try

  Dim xr As New XmlTextReader(s)
  Dim rawData As XDocument = XDocument.Load(xr)

  ' поместить данные из каждого элемента status
  ' в новый объект Tweet
```

```
    Dim query = From entry In _
                rawData.Descendants("statuses").Descendants("status") _
                Select New Tweet With { _
                .UserName = entry.Element("user").Element("screen_name").Value, _
                .Text = entry.Element("text").Value, _
                .DateCreated = DateTime.ParseExact( _
                        entry.Element("created_at").Value, _
                        "ddd MMM dd HH:mm:ss zzzz yyyy", _
                        CultureInfo.GetCultureInfoByIetfLanguageTag("en-us"), _
                        DateTimeStyles.AllowWhiteSpaces)}
End Function
```

В первой части метода создается объект `WebClient`, в нем устанавливаются имя и пароль, переданные конструктору, после чего открывается указанный Uri. Если возникло исключение, мы определяем причину ошибки и возбуждаем соответствующее исключение для вызывающей программы. Twitter возвращает ошибку с кодом 400, если превышена частота обращений.

Во второй части мы преобразуем поток, полученный от объекта `WebClient`, в объект `XDocument`, представляющий XML-документ. Затем, применяя технологию *LINQ to XML*, мы формируем запрос к XML-данным и создаем список `List` объектов типа `Tweet`, заполняя свойства `UserName`, `Text` и `DateCreated` данными, которые вернула служба.

Реализация голосования

Научившись посылать запросы службе Twitter и получать от нее данные, мы можем приступить к реализации самого опроса и сбору результатов. Добавьте в папку *App_Code* новый класс `Poll`.

Для начала подготовим предложения using, переменные-члены и заготовку класса. В переменных-членах будут храниться экземпляр класса `TwitterService`, допустимые в опросе ответы, голоса других пользователей и две временные метки: одна показывает, когда мы в последний раз обращались к Twitter, а другая позволяет отличить новые голоса от старых. Конструктор читает имя и пароль пользователя из файла *web.config* (мы приведем его позже) и создает объект `TwitterService`. Все это показано в примерах 6.8 и 6.9.

Пример 6.8. Заготовка класса Poll (версия на C#)

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Configuration;

public class Poll
{
  // Наша обертка Twitter API
  private TwitterService twitter;
```

```
// голоса других пользователей
private List<string> votes { get; set; }

// допустимые ответы в опросе
private HashSet<string> validResponses { get; set; }

// когда в последний раз обновляли информацию
private DateTime lastUpdatedVotes;
private DateTime lastUpdatedPolls;

// сопоставление каждому варианту ответа процентной доли
Dictionary<string, double> responses;

public Poll()
{
  validResponses = new HashSet<string>();
  votes = new List<string>();

  twitter = new TwitterService(
     ConfigurationManager.AppSettings["TwitterUserName"],
     ConfigurationManager.AppSettings["TwitterPassword"]);
```

Пример 6.9. Заготовка класса Poll (версия на VB)

```
Imports Microsoft.VisualBasic
Imports System
Imports System.Collections.Generic
Imports System.Linq
Imports System.Configuration

Public Class Poll
  ' Наша обертка Twitter API
  Private twitter As TwitterService

  ' голоса других пользователей
  Private privatevotes As List(Of String)
  Private Property votes() As List(Of String)
    Get
      Return privatevotes
    End Get
    Set(ByVal value As List(Of String))
      privatevotes = value
    End Set
  End Property

  ' допустимые ответы в опросе
  Private privatevalidResponses As HashSet(Of String)
  Private Property validResponses() As HashSet(Of String)
    Get
      Return privatevalidResponses
    End Get
    Set(ByVal value As HashSet(Of String))
      privatevalidResponses = value
    End Set
  End Property
```

```
    ' когда в последний раз обновляли информацию
    Private lastUpdatedVotes As DateTime
    Private lastUpdatedPolls As DateTime

    ' сопоставление каждому варианту ответа процентной доли
    Private responses As Dictionary(Of String, Double)

    Public Sub New()
      validResponses = New HashSet(Of String)()
      votes = New List(Of String)()

      twitter = New TwitterService( _
        ConfigurationManager.AppSettings("TwitterUserName"), _
        ConfigurationManager.AppSettings("TwitterPassword"))
    End Sub
  End Class
```

Далее нам нужен метод для поиска «команд опроса» в списке собственных твитов. Для создания опроса пользователь должен отправить твит в формате `poll` ***pollparameters***, где `pollparameters` – либо два числа, разделенные черточкой (например, «1-10»), либо список допустимых ответов через запятую (например, «A,B,C,D»). Если заданы два числа, то предполагается, что допустимы любые ответы в указанном диапазоне, включая и границы. Например, на опрос с параметрами 1-10 можно давать в качестве ответа любое число от 1 до 10 включительно.

Метод `UpdatePolls` извлекает «временной отрезок» пользователя. Проще говоря, это все отправленные пользователем твиты. Метод ищет в полученном списке объектов `Tweet` те, что имеют описанный выше формат. Обнаружив подходящий твит, метод обрабатывает список допустимых ответов, обращаясь к методу `ProcessParameterList`, и сохраняет их для последующего использования. Закончив обработку списка, мы сохраняем время последнего сообщения, так чтобы при следующем получении списка можно было пропустить старые записи. Код представлен в примерах 6.10 и 6.11.

Пример 6.10. Методы UpdatePolls и ProcessParameterList (версия на C#)

```
private void UpdatePolls()
{
  // получаем все твиты пользователя
  List<Tweet> tweets = twitter.GetUserTimeline();

  if (tweets == null || tweets.Count <= 0)
    return;

  foreach (Tweet tweet in tweets)
  {
    // если твит новый
    if (tweet.DateCreated.HasValue && tweet.DateCreated > lastUpdatedPolls)
    {
      // ищем "параметры опроса"
      string[] breakdown = tweet.Text.ToLower().Split(' ');
```

```
      // если это команда poll, создаем новый опрос
      if (breakdown.Length == 2 && breakdown[0].ToLower() == "poll")
      {
        ProcessParameterList(breakdown[1]);
        break;
      }
    }
  }

  if (tweets.Count > 0)
    lastUpdatedPolls = tweets[0].DateCreated.Value;
}

private void ProcessParameterList(string parameterList)
{
  // сейчас поддерживаются
  // 1-10 \d-\d
  // a,b,c,d через запятую.

  // числа от X до Y
  if(parameterList.Contains('-'))
  {
    string[] items = parameterList.Split('-');
    if (items.Length == 2)
    {
      // если есть черточка, предполагаем, что это два
      // числа: нижнее - верхнее
      int start = int.Parse(items[0]);
      int end = int.Parse(items[1]);

      for (int i = start; i <= end; i++)
        validResponses.Add(i.ToString());
    }
  }
  // список вариантов через запятую
  else if(parameterList.Contains(','))
  {
    string[] items = parameterList.Split(',');
    foreach (string item in items)
      validResponses.Add(item.Trim());
  }
}
```

Пример 6.11. Методы UpdatePolls и ProcessParameterList (версия на VB)

```
Private Sub UpdatePolls()
  ' получаем все твиты пользователя
  Dim tweets As List(Of Tweet) = twitter.GetUserTimeline()

  If tweets Is Nothing OrElse tweets.Count <= 0 Then
    Return
  End If

  For Each tweet As Tweet In tweets
    ' если твит новый
```

```
    If tweet.DateCreated.HasValue AndAlso _
            tweet.DateCreated > lastUpdatedPolls Then
       ' ищем "параметры опроса"
       Dim breakdown() As String = tweet.Text.ToLower().Split(" "c)

       ' если это команда poll, создаем новый опрос
       If breakdown.Length = 2 AndAlso breakdown(0).ToLower() = "poll" Then
         ProcessParameterList(breakdown(1))
         Exit For
         End If
      End If
  Next tweet

    If tweets.Count > 0 Then
        lastUpdatedPolls = tweets(0).DateCreated.Value
    End If
End Sub

Private Sub ProcessParameterList(ByVal parameterList As String)
  ' сейчас поддерживаются
  ' 1-10 \d-\d
  ' a,b,c,d через запятую.

  ' числа от X до Y
  If parameterList.Contains("-"c) Then
    Dim items() As String = parameterList.Split("-"c)
    If items.Length = 2 Then
      ' если есть черточка, предполагаем, что это два
      ' числа: нижнее - верхнее
      Dim start As Integer = Integer.Parse(items(0))
      Dim [end] As Integer = Integer.Parse(items(1))

      For i As Integer = start To [end]
        validResponses.Add(i.ToString())
      Next i
    End If

  ' список вариантов через запятую
  ElseIf parameterList.Contains(","c) Then
    Dim items() As String = parameterList.Split(","c)
    For Each item As String In items
      validResponses.Add(item.Trim())
    Next item
  End If
End Sub
```

Итак, опрос создан, и нам интересно узнать, как другие пользователи проголосовали. Метод `UpdateVotes` извлекает ответы, посланные текущему пользователю, и ищет в списке те, что содержат допустимый вариант ответа на опрос (что является допустимым, мы уже обсудили выше). Найденные допустимые ответы добавляются в список голосов для последующего оформления в виде таблицы. Как и в случае с методом `UpdatePolls`, мы запоминаем время последнего ответа, чтобы в следующий раз обрабатывать только новые ответы. Попутно добавим

вспомогательный метод `UpdateAll`**, который будет вызывать оба вышеупомянутых метода, чтобы найти последний по времени опрос и обновить результаты голосования (см. примеры 6.12 и 6.13).**

Пример 6.12. Методы UpdateVotes и UpdateAll (версия на C#)

```
private void UpdateVotes()
{
  // получить все ответы, адресованные текущему пользователю
  List<Tweet> tweets = twitter.GetReplies();

  // ответ приходит в виде @username Vote
  foreach (Tweet tweet in tweets)
  {
    // если мы уже видели его раньше, пропускаем
    if (tweet.DateCreated.HasValue && tweet.DateCreated > lastUpdatedVotes)
    {
      // разбить на части
      string[] tweetSplit = tweet.Text.ToLower().Split(' ');

      // если количество отрезков правильное и ответ содержит
      // допустимый вариант, добавляем в список голосов
      if (tweetSplit.Length == 2 &&  validResponses.Contains(tweetSplit[1]))
        votes.Add(tweetSplit[1]);
    }
  }

  // если что-то обновлялось, запомнить время обновления
  if(tweets.Count > 0)
    lastUpdatedVotes = tweets[0].DateCreated.Value;
}

public void UpdateAll()
{
  UpdatePolls();
  UpdateVotes();
}
```

Пример 6.13. Методы UpdateVotes и UpdateAll (версия на VB)

```
Private Sub UpdateVotes()
  ' получить все ответы, адресованные текущему пользователю
  Dim tweets As List(Of Tweet) = twitter.GetReplies()

  ' ответ приходит в виде @username Vote
  For Each tweet As Tweet In tweets
    ' если мы уже видели его раньше, пропускаем
    If tweet.DateCreated.HasValue AndAlso _
            tweet.DateCreated > lastUpdatedVotes Then
      ' разбить на части
      Dim tweetSplit() As String = tweet.Text.ToLower().Split(" "c)

      ' если количество отрезков правильное и ответ содержит
      ' допустимый вариант, добавляем в список голосов
      If tweetSplit.Length = 2 AndAlso _
```

```
            validResponses.Contains(tweetSplit(1)) Then
          votes.Add(tweetSplit(1))
        End If
      End If
    Next tweet

    ' если что-то обновлялось, запомнить время обновления
    If tweets.Count > 0 Then
      lastUpdatedVotes = tweets(0).DateCreated.Value
    End If
  End Sub

  Public Sub UpdateAll()
    UpdatePolls()
    UpdateVotes()
  End Sub
```

Теперь у нас есть и опрос, и результаты. Настало время оформить их в виде таблицы. Метод `GetPollResults` **создает объект** `Dictionary`**, в котором каждому допустимому ответу сопоставлена процентная доля проголосовавших за него. Голоса подсчитываются и возвращаются вызывающей программе для последующего отображения. Кроме того, мы добавим еще одно свойство** `TotalVotes`**, в котором будет храниться общее число поданных голосов (см. примеры 6.14 и 6.15).**

Пример 6.14. Метод GetPollResults и свойство TotalVotes (версия на C#)

```
public Dictionary<string, double> GetPollResults()
{
  // сопоставляем варианту ответа процентную долю
  Dictionary<string, double> returnResults =
      new Dictionary<string, double>(validResponses.Count);

  // инициализируем словарь допустимыми ответами
  foreach (string param in validResponses)
      returnResults[param] = 0;

  // подводим итоги голосования
  foreach (string vote in votes)
  {
    if (returnResults.Keys.Contains(vote))
      returnResults[vote] += 1;
  }

  if (votes.Count > 0)
  {
    // вычисляем процентную долю
    string[] keys = returnResults.Keys.ToArray();
    foreach (string key in keys)
      returnResults[key] = (returnResults[key] / votes.Count) * 100;
  }

  return returnResults;
}
```

```
public int TotalVotes
{
  get { return votes.Count; }
}
```

Пример 6.15. Метод GetPollResults и свойство TotalVotes (версия на VB)

```
Public Function GetPollResults() As Dictionary(Of String, Double)
  ' сопоставляем варианту ответа процентную долю
  Dim returnResults As Dictionary(Of String, Double) = _
      New Dictionary(Of String, Double)(validResponses.Count)

  ' инициализируем словарь допустимыми ответами
  For Each param As String In validResponses
    returnResults(param) = 0
  Next param

  ' подводим итоги голосования
  For Each vote As String In votes
    If returnResults.Keys.Contains(vote) Then
      returnResults(vote) += 1
    End If
  Next vote

  If votes.Count > 0 Then
    ' вычисляем процентную долю
    Dim keys() As String = returnResults.Keys.ToArray()
    For Each key As String In keys
      returnResults(key) = (returnResults(key) / votes.Count) * 100
    Next key
  End If

  Return returnResults
End Function

Public ReadOnly Property TotalVotes() As Integer
  Get
    Return votes.Count
  End Get
End Property
```

Отображение результатов с помощью Popfly

Теперь, когда мы умеем создавать опрос, получать ответы и подводить итоги, нужно как-то вывести результаты на экран. Для этого мы создадим Popfly-компонент и внедрим его в веб-страницу.

Сайт Popfly (*http://www.popfly.com/*) позволяет пользователям, не имеющим опыта программирования, создавать *mashup*-приложения. Так называются веб-приложения, которые собирают данные из разных источников и каким-то образом их комбинируют, получая новое приложение.

В главном окне Popfly вы можете перетаскивать крупные блоки на поверхность конструирования, конфигурировать их для решения определенных задач, а затем соединять входы и выходы блоков, организуя поток и трансформации данных, необходимые для достижения конечной цели.

Компонент, который мы собираемся построить, находится в общем пользовании по адресу *http://www.popfly.com/users/c4fbook/TwitterVoteGraph*, так что можете применять его напрямую. А если хотите создать с нуля Popfly-компонент для отображения результатов, следуйте приведенным ниже инструкциям. Конечный результат будет выглядеть, как показано на рис. 6.4.

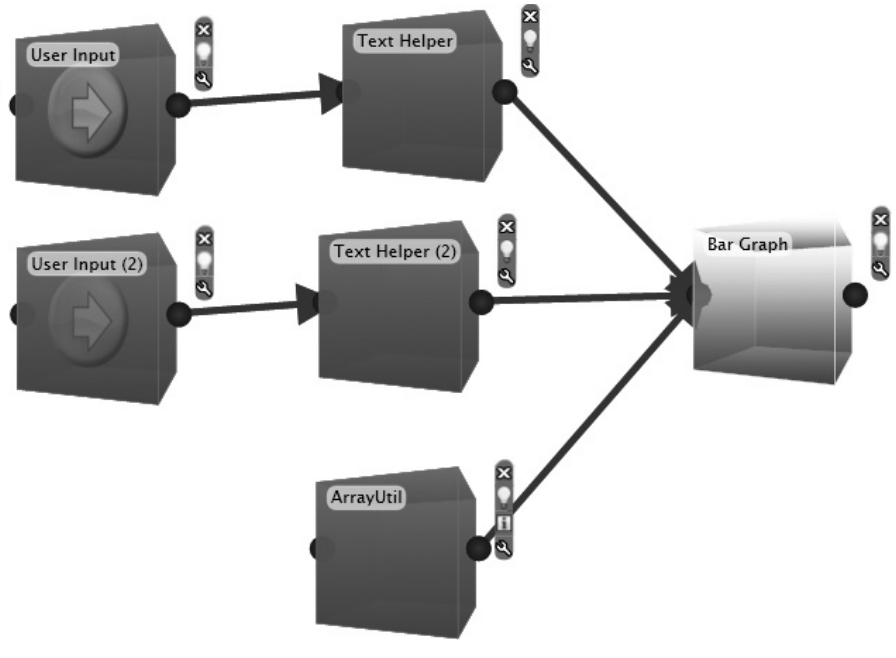

Рис. 6.4. *Окончательный вид mashup-компонента для TwitterVoteGraph*

Чтобы сделать такую штуку, для начала создайте новое mashup-приложение в Popfly. Найдите на левой панели Blocks блок *User Input* и перетащите два таких блока в область конструирования. То же самое проделайте для блоков *Text Helper*, *ArrayUtil* и *Bar Graph*. Соедините их, как показано на рис. 6.4.

Далее необходимо сконфигурировать оба блока *User Input*, так чтобы они извлекали надписи и значения из строки запроса в URL. Щелкните по значку с изображением гаечного ключа справа от каждого блока *User Input* и сконфигурируйте их, как показано на рис. 6.5 и 6.6.

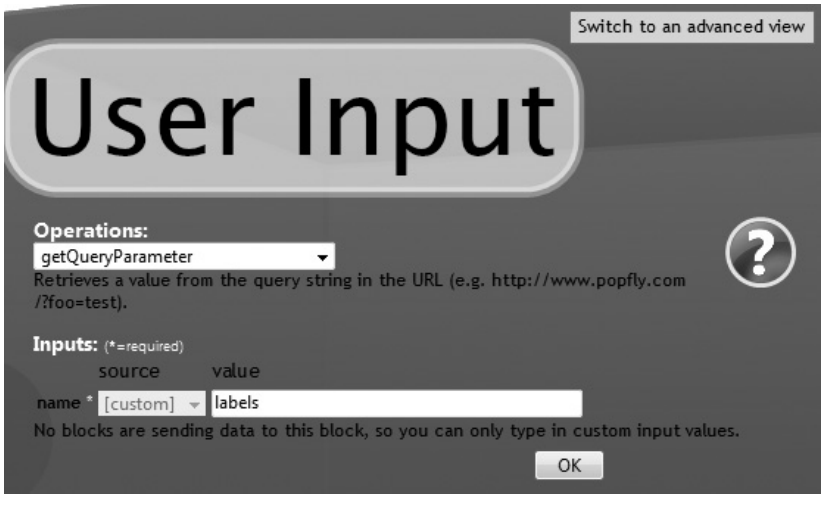

Рис. 6.5. *Блок User Input для извлечения названий*

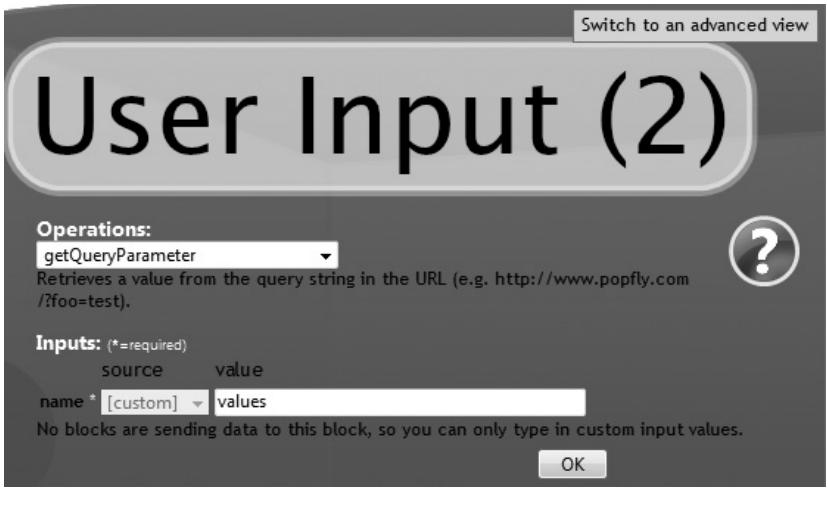

Рис. 6.6. *Блок User Input для извлечения значений*

Далее сконфигурируем блоки *Text Helper* так, чтобы выделить компоненты надписей и значений, считая, что они разделены знаком вертикальной черты (|). Конфигурационные параметры должны быть заданы, как показано на рис. 6.7 и 6.8.

Блок *ArrayUtil* используется для хранения массива цветов, передаваемого в блок *BarGraph* для раскраски столбиков диаграммы. Сконфигурируйте его, как показано на рис. 6.9.

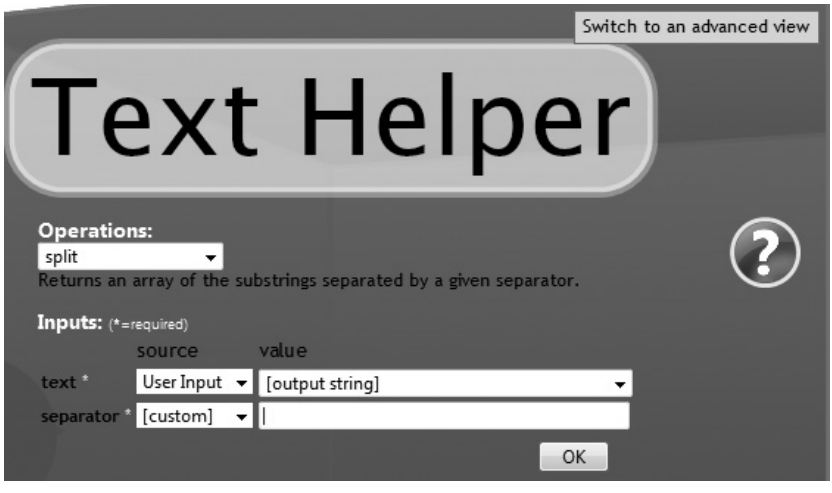

Рис. 6.7. *Блок Text Helper для названий*

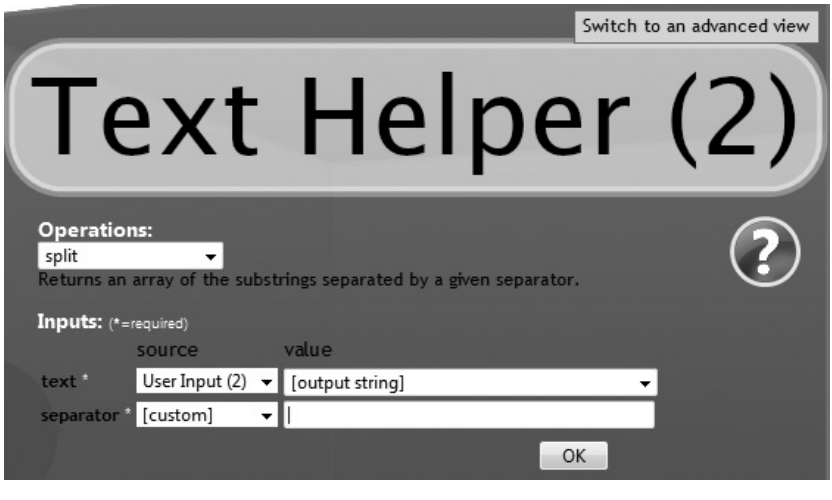

Рис. 6.8. *Блок Text Helper для значений*

И напоследок сконфигурируем блок *Bar Graph*, пользуясь выходной информацией от предыдущих блоков (рис. 6.10).

Внедрение диаграммы с сайта Popfly

Имея Popfly-компонент (общедоступный или тот, что вы создали сами), можно внедрить его в свою веб-страницу и отобразить результаты. HTML-разметка страницы *default.aspx* совсем проста. Нужно лишь

Рис. 6.9. *Конфигурация блока ArrayUtil*

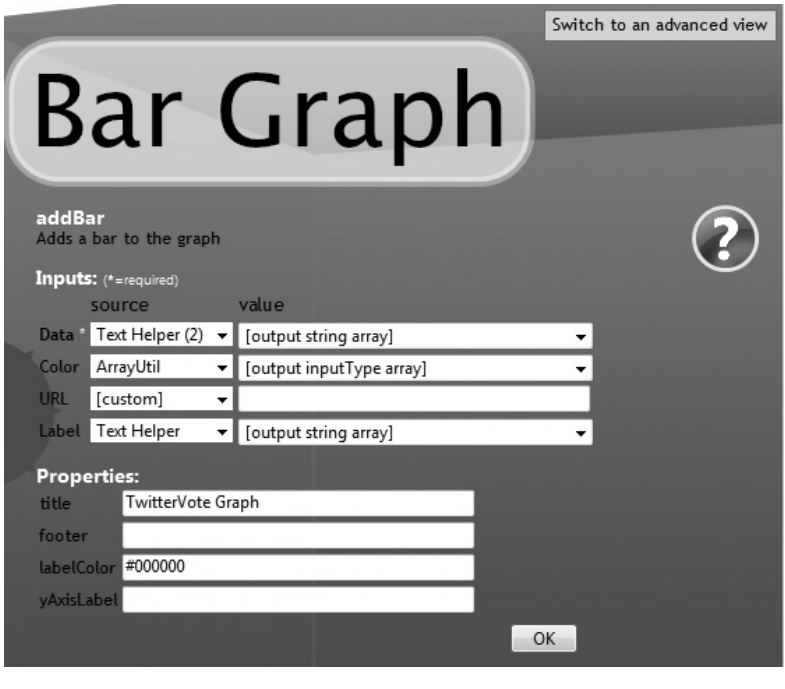

Рис. 6.10. *Конфигурация блока Bar Graph*

вставить тег `<IFRAME>`, в котором будет находиться импортируемое с сайта Popfly содержимое, несколько меток для вывода общего количества голосов и кнопку для обновления графика (см. примеры 6.16 и 6.17).

Пример 6.16. HTML-разметка страницы Default.aspx (версия на C#)

```
<%@ Page Language="C#" AutoEventWireup="true" CodeFile="Default.aspx.cs"
Inherits="Default" %>
<html>
<head runat="server">
  <title>TwitterVote</title>
</head>
<body style="font-family:Arial">
  <form id="form1" runat="server">
  <div ID="ResultsData" runat="server" />
  <p/>
  <asp:Label ID="TotalVotes" runat="server" Text="Total Votes: "/>
  <asp:Label ID="Total" runat="server" Text="Label" />
  <p/>
  <asp:Button ID="UpdateResults" runat="server" onclick="UpdateResults_Click"
    Text="Update Results" />
  </form>
</body>
</html>
```

Пример 6.17. HTML-разметка страницы Default.aspx (версия на VB)

```
<%@ Page Language="vb" AutoEventWireup="true" CodeFile="Default.aspx.vb"
Inherits="Default" %>
<html>
<head runat="server">
  <title>TwitterVote</title>
</head>
<body style="font-family:Arial">
  <form id="form1" runat="server">
  <div ID="ResultsData" runat="server" />
  <p/>
  <asp:Label ID="TotalVotes" runat="server" Text="Total Votes: "/>
  <asp:Label ID="Total" runat="server" Text="Label" />
  <p/>
  <asp:Button ID="UpdateResults" runat="server" onclick="UpdateResults_Click"
    Text="Update Results" />
  </form>
</body>
</html>
```

Код программной логики этой ASP.NET-страницы тоже не вызывает сложностей. Сразу после загрузки страницы мы создаем объект `Poll`. Если он уже находится в кэше, то возьмем его оттуда, чтобы сократить время обработки, в противном случае создадим с нуля. Код приведен в примерах 6.18 и 6.19.

Пример 6.18. Код программной логики страницы Default.aspx (версия на C#)

```
using System;
using System.Collections.Generic;
using System.Text;
```

```
using System.Web;

public partial class Default : System.Web.UI.Page
{
  private Poll poll;

  protected void Page_Load(object sender, EventArgs e)
  {
    // получаем объект Poll
    poll = GetPoll();

    // обновляем при первой загрузке
    if(!Page.IsPostBack)
    {
      poll.UpdateAll();
      DrawPopflyResults();
    }
  }

  private Poll GetPoll()
  {
    // если объект Poll находится в кэше, берем оттуда
    // иначе создаем новый
    Poll poll = (Poll)HttpRuntime.Cache["poll"];
    if (poll == null)
    {
      poll = new Poll();
      HttpRuntime.Cache["poll"] = poll;
    }
    return poll;
  }
}
```

Пример 6.19. Код программной логики страницы Default.aspx (версия на VB)

```
Imports System
Imports System.Collections.Generic
Imports System.Text
Imports System.Web

Partial Public Class [Default]
  Inherits System.Web.UI.Page
  Private poll As Poll
  Protected Sub Page_Load(ByVal sender As Object, ByVal e As EventArgs)
    ' получаем объект Poll
    poll = GetPoll()

    ' обновляем при первой загрузке
    If (Not Page.IsPostBack) Then
      poll.UpdateAll()
      DrawPopflyResults()
    End If
  End Sub

  Private Function GetPoll() As Poll
```

```
    ' если объект Poll находится в кэше, берем оттуда
    ' иначе создаем новый
    Dim poll As Poll = CType(HttpRuntime.Cache("poll"), Poll)
    If poll Is Nothing Then
      poll = New Poll()
      HttpRuntime.Cache("poll") = poll
    End If
    Return poll
  End Function
End Class
```

Далее нужно написать метод, который будет получать результаты, представлять их в формате, ожидаемом Popfly, и вставлять окончательный URL в тег `<IFRAME>`, чтобы отобразить результаты на странице. Popfly-компонент ожидает найти в строке запроса надписи и значения. Те и другие должны быть представлены в виде списка с разделителем |. Код метода `DrawPopflyResults` приведен в примерах 6.20 и 6.21.

Пример 6.20. Метод DrawPopflyResults (версия на C#)

```
private void DrawPopflyResults()
{
  // получаем результаты в виде таблицы
  Dictionary<string, double> results = poll.GetPollResults();
  StringBuilder labels = new StringBuilder();
  StringBuilder values = new StringBuilder();

  // создаем списки надписей и значений для построения диаграммы
  // элементы разделяем знаком |
  foreach(KeyValuePair<string, double> result in results)
  {
    labels.AppendFormat("{0}|", result.Key);
    values.AppendFormat("{0}|", result.Value);
  }

  // создание
  string graphUrl =  string.Format(@"http://www.popfly.com/users/c4fbook/?
TwitterVoteGraph.content?labels={0}&values={1}", labels, values);
  ResultsData.InnerHtml = string.Format(@"<iframe allowtransparency='true' ?
frameborder='no' src='{0}' style='width: 800px;
height: 600px'></iframe>", graphUrl);

  Total.Text = poll.TotalVotes.ToString();
}
```

Пример 6.21. Метод DrawPopflyResults (версия на VB)

```
Private Sub DrawPopflyResults()
  ' получаем результаты в виде таблицы
  Dim results As Dictionary(Of String, Double) = poll.GetPollResults()
  Dim labels As New StringBuilder()
  Dim values As New StringBuilder()

  ' создаем списки надписей и значений для построения диаграммы
```

```
    ' элементы разделяем знаком |
    For Each result As KeyValuePair(Of String, Double) In results
      labels.AppendFormat("{0}|", result.Key)
      values.AppendFormat("{0}|", result.Value)
    Next result

    ' создание
    Dim graphUrl As String = String.Format("http://www.popfly.com/users/c4fbook/?
TwitterVoteGraph.content?labels={0}&values={1}", labels, values)
    ResultsData.InnerHtml = String.Format("<iframe allowtransparency='true' ?
frameborder='no' src='{0}' style='width: 800px; height: 600px'></iframe>",
    graphUrl)

    Total.Text = poll.TotalVotes.ToString()
End Sub
```

Наконец, напишем обработчик нажатия на кнопку, в котором обновим данные об опросе и заново выведем результаты (см. примеры 6.22 и 6.23).

Пример 6.22. Метод UpdateResults_Click (версия на C#)

```
protected void UpdateResults_Click(object sender, EventArgs e)
{
  poll.UpdateAll();
  DrawPopflyResults();
}
```

Пример 6.23. Метод UpdateResults_Click (версия на VB)

```
Protected Sub UpdateResults_Click(ByVal sender As Object, ByVal e As
EventArgs)
  poll.UpdateAll()
  DrawPopflyResults()
End Sub
```

Запуск приложения

Последний шаг – прописать в файле *web.config* имя и пароль пользователя Twitter. Открыв существующий файл *web.config*, вы увидите ближе к началу строку `<appSettings/>`. Замените ее параметрами приложения, как показано в примере 6.24, подставив вместо строк «username» и «password» свои имя и пароль пользователя Twitter.

Пример 6.24. Секция appSettings в файле web.config

```
<appSettings>
  <add key="TwitterUserName" value="username"/>
  <add key="TwitterPassword" value="password"/>
</appSettings>
```

Теперь нажмите клавишу F5 или выберите из меню команду Debug→ Start Debugging, чтобы запустить приложение в режиме отладки. Будет создан локальный веб-сервер, и приложение начнет искать последнюю

команду `poll` и результаты соответствующего опроса. Создайте свой опрос, предложите народу принять в нем участие и нажмите кнопку Update Results, чтобы получить итоги голосования.

Однако не забывайте, что Twitter ограничивает частоту обращений, поэтому не нажимайте кнопку слишком часто, иначе вас на какое-то время заблокируют.

Пример опроса и результатов голосования

Давайте взглянем на пример опроса и голосования по нему. На рис. 6.11 показано, как «твитерянин» по имени C4FBook задает вопрос: «Ваша любимая игровая приставка?». Пользователь создал опрос, отправив команду `poll wii,ps3,xbox360`, которая начинает опрос и задает допустимые ответы «Wii, Ps3 и Xbox360». Видно, что ответили четыре пользователя.

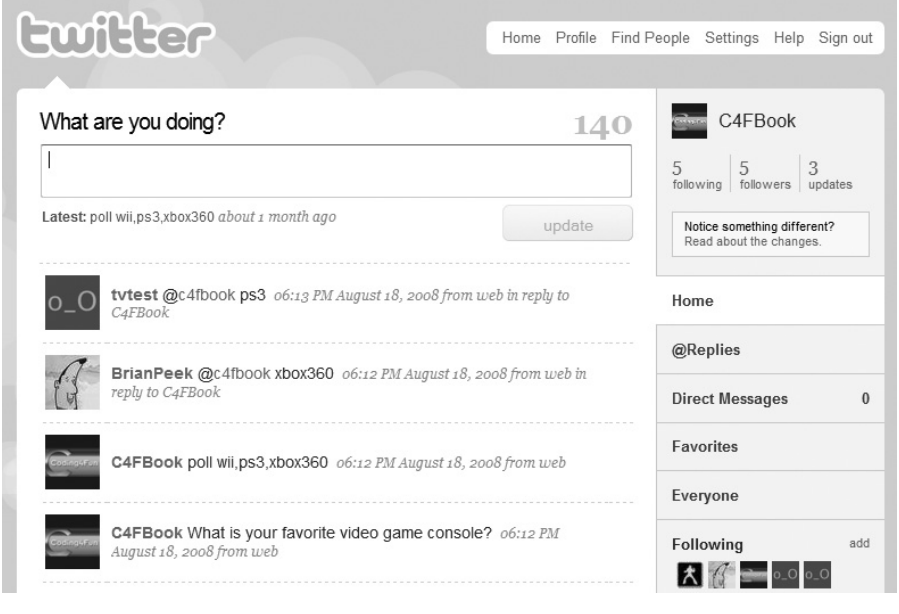

Рис. 6.11. *Создание опроса и ответы на него*

Имея опрос и проголосовавших, мы можем запустить приложение TwitterVote и с помощью Popfly увидеть гистограмму с итогами голосования (рис. 6.12).

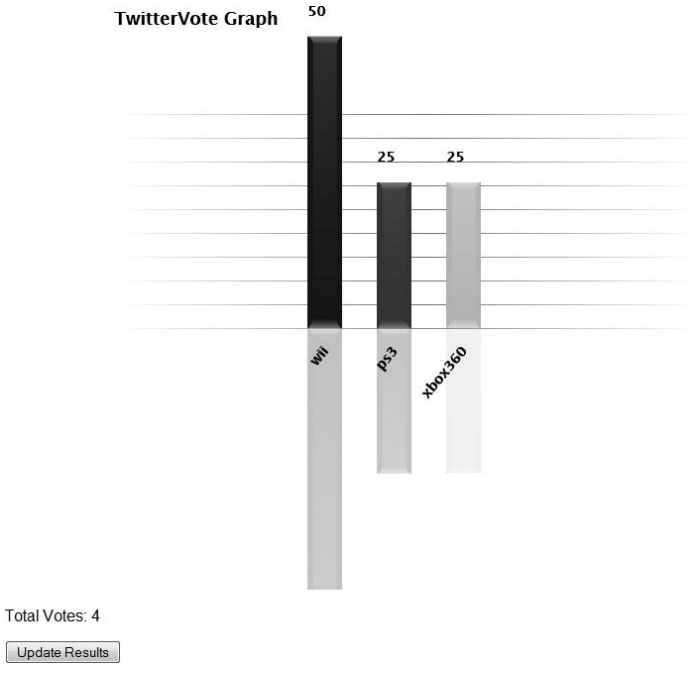

Рис. 6.12. *Диаграмма итогов голосования, полученная с помощью Popfly*

Заключительные замечания

Twitter – чрезвычайно популярная веб-служба с не слишком сложным API. Приложение TwitterVote – лишь один простой пример использования этого API. Его можно обобщить, добавив возможности, для которых необходимы другие предлагаемые службой методы. Например, можно написать викторину. Испытайте это приложение в деле – заведите опрос, в котором спросите у своих друзей, что они хотели бы увидеть, а потом реализуйте их пожелания!

7

WHSMail: надстройка над Windows Home Server для чтения почты в Outlook

Автор	Брайан Пик
Сложность	Средняя
Необходимое время	3–6 часов
Стоимость	Бесплатно
Программное обеспечение	Windows Home Server, Microsoft Outlook 2007 (не Outlook Express/Windows Mail), Visual Basic 2008 Express или Visual C# 2008 Express, Visual Web Developer 2008 Express, исполняющая среда .NET Framework 3.5 и Outlook/Office Primary Interop Assemblies (подробнее об этом ниже)
Оборудование	Не требуется
Адрес в Интернете	*http://www.c4fbook.com/WHSMail*

В этой главе мы покажем, как можно создать с помощью ASP.NET вебсайт, который, будучи установлен в системе Windows Home Server (WHS), позволит просматривать хранилище сообщений Outlook с любого компьютера в сети, на котором настроен Outlook.

Краткий обзор

Windows Home Server – это продукт корпорации Microsoft, который позволяет домашним пользователям управлять и предоставлять в общее пользование разнообразные данные, в том числе фотографии, документы, видео и музыку. Кроме того, он позволяет без особого труда

хранить резервные копии всех компьютеров в домашней сети на центральном сервере. Windows Home Server допускает расширение за счет надстроек, позволяющих реализовать новые интересные функции в дополнение к стандартным.

Функция, которую WHS не предоставляет и которой мне не хватает, – это возможность в любой момент просматривать хранилище электронной почты в Outlook из сети. У меня есть шесть или семь почтовых ящиков, которые настроены так, что Outlook забирает из них почту по протоколу POP 3. Большинство этих ящиков не поддерживают протокол IMAP и не имеют веб-интерфейса. Поэтому Outlook обычно весь день открыт и проверяет наличие новой почты. Вся моя старая корреспонденция хранится там же. Если я нахожусь не дома, то зачастую неудобно открывать удаленный рабочий стол на машине, где работает Outlook, только чтобы прочитать почту. Хорошо было бы иметь веб-версию папок Outlook, чтобы для просмотра всей своей почты мне достаточно было зайти на веб-сервер, организованный на домашнем компьютере. В дистрибутив Windows Home Server включен веб-сервер Internet Information Services 6 (IIS6), так что создать новое веб-приложение не составляет никакого труда.

Установка и конфигурирование

Тут придется немного потрудиться. На машину, на которой происходит разработка, необходимо установить Outlook 2007. Совсем необязательно ставить его на ту же машину, где находится хранилище почты, хотя это было бы удобно. Затем следует установить *Primary Interop Assemblies* для Office. Для этого в программе установки ОС выберите из списка подкомпонентов Microsoft Office Outlook пункт *.NET Programmability Support*, как показано на рис. 7.1.

Архитектура

Выбранная нами архитектура очень похожа на архитектуру N-уровневого приложения, когда приложение-клиент соединяется с сервером для получения данных или выполнения какой-то операции. Машину, на которой работает Outlook 2007 и находится хранилище данных, можно считать сервером. Там будет работать наш серверный процесс, представляющий несколько методов с помощью платформы Windows Communication Foundation (WCF). К этим методам будет обращаться ASP.NET-приложение, работающее на машине под управлением Windows Home Server.

Создание хост-приложения

Начнем с создания хост-приложения. Оно будет работать на компьютере, где установлен Outlook и находятся почтовые сообщения. Визу-

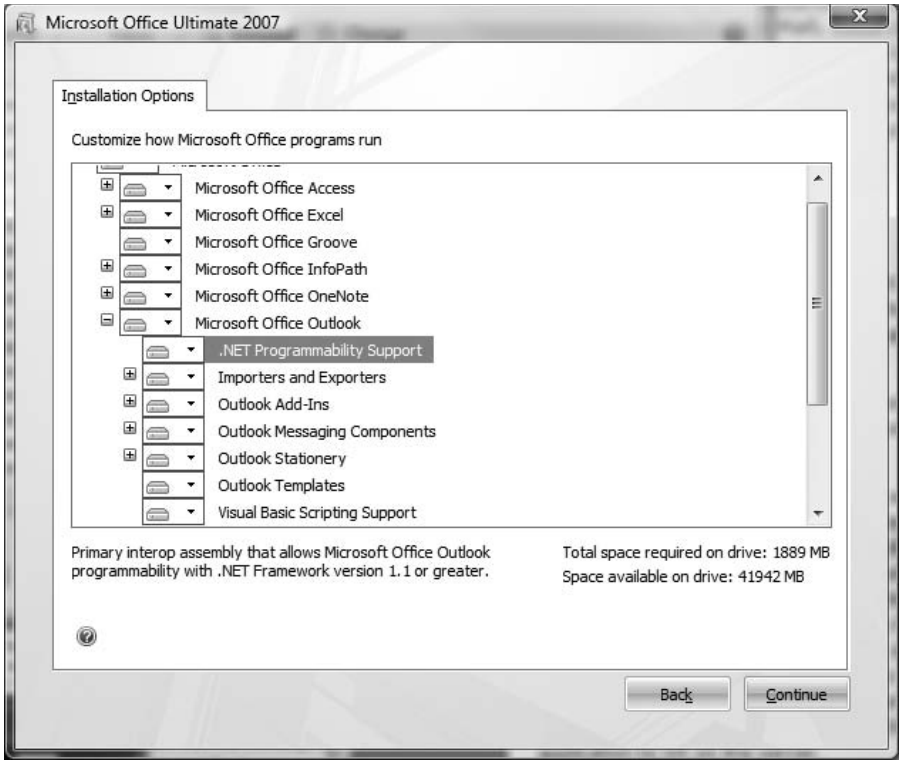

Рис. 7.1. *Установка Outlook 2007, .NET Programmability Support*

ально приложение будет располагаться в области уведомлений, рядом с часами Windows.

Откройте Visual C# 2008 Express и создайте проект типа Windows Application с именем `WHSMailHost`. Переименуйте созданный по умолчанию файл *Form1.cs* (или *.vb*) в *frmMain.cs* (или *.vb*). Дважды щелкните по этому файлу на панели Solution Explorer, чтобы открыть область проектирования.

Для того чтобы приложение запускалось в области уведомлений, перетащите элемент управления `Notify Icon` из инструментария в область проектирования. Назовите его `niIcon`, затем перетащите в область проектирования элемент управления `ContextMenuStrip` и назовите его `cmsMenu` (рис. 7.2).

Установите свойства элемента `niIcon`, как показано в табл. 7.1 и на рис. 7.3.

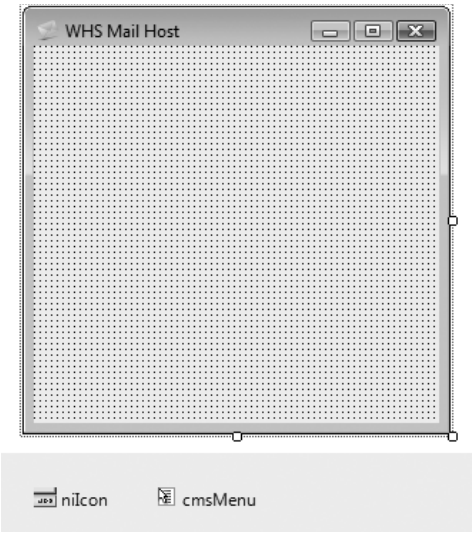

***Рис. 7.2.** Форма после добавления NotifyIcon и ContextMenuStrip*

Таблица 7.1. Свойства niIcon

Свойство	Значение
Text	WHS Mail Host
ContextMenuStrip	cmsMenu
Visible	True

Щелкните правой кнопкой мыши по элементу cmsMenu и выберите из меню команду Edit Items... – откроется окно редактора пунктов меню, показанное на рис. 7.4.

Добавьте в меню один пункт mnuExit и установите его свойство Text равным &Exit, как показано на рис. 7.5.

Нажмите OK, чтобы закрыть диалоговое окно. Вернувшись в область проектирования, вы увидите, что в верхней части формы появилось меню. Дважды щелкните по только что добавленному пункту Exit, чтобы связать с ним событие по умолчанию Click. В обработчике события Click просто закройте форму, как показано в примерах 7.1 и 7.2.

Пример 7.1. Обработчик события mnuExit_Click event (версия на C#)

```
private void mnuExit_Click(object sender, EventArgs e)
{
  // выход из приложения
  this.Close();
}
```

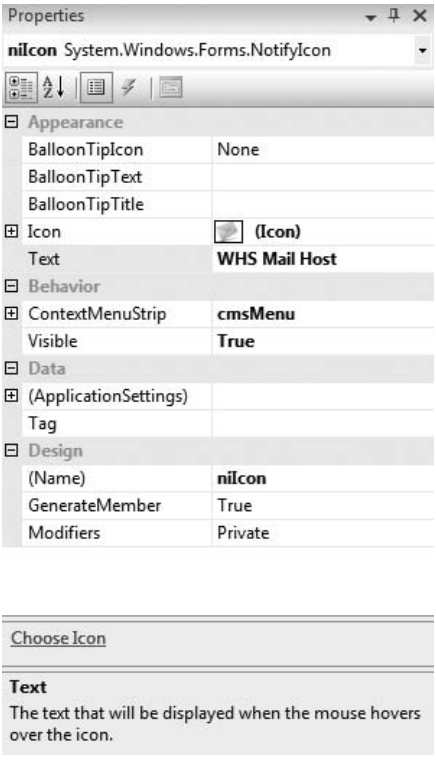

Рис. 7.3. *Свойства niIcon*

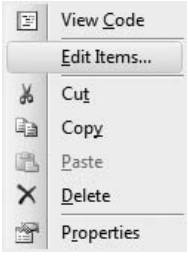

Рис. 7.4. *Пункт меню Edit Items...*

Пример 7.2. Обработчик события mnuExit_Click event (версия на VB)

```
Private Sub mnuExit_Click(ByVal sender As Object, ByVal e As EventArgs) _
  Handles mnuExit.Click
  ' выход из приложения
  Me.Close()
End Sub
```

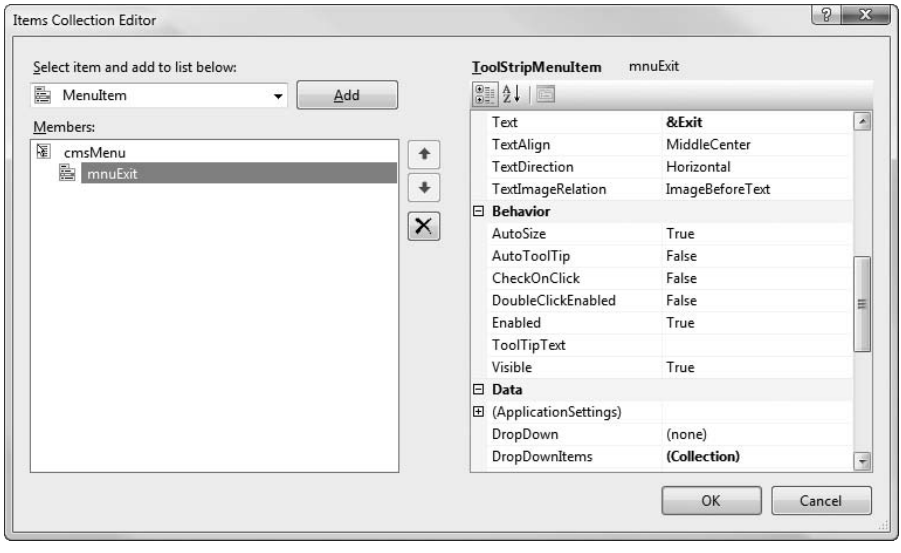

Рис. 7.5. *Редактор пунктов меню*

Затем добавьте события `Resize`**,** `Load` **и** `FormClosing` **в форму** `frmMain`**, выбрав эти события на панели Event Property (рис. 7.6). Реализуйте обработчики, как показано в примерах 7.3 и 7.4.**

Пример 7.3. Обработчики событий Resize, Load и FormClosing (версия на C#)

```
private void frmMain_Resize(object sender, EventArgs e)
{
  // скрыть форму
  this.Hide();
}

private void frmMain_Load(object sender, EventArgs e)
{
  // запустить WCF-службу
  MyServiceHost.StartService();
}

private void frmMain_FormClosing(object sender, FormClosingEventArgs e)
{
  // остановить WCF-службу
  MyServiceHost.StopService();
}
```

Пример 7.4. Обработчики событий Resize, Load и FormClosing (версия на VB)

```
Private Sub frmMain_Resize(ByVal sender As Object, ByVal e As EventArgs) _
  Handles MyBase.Resize
  ' скрыть форму
  Me.Hide()
End Sub
```

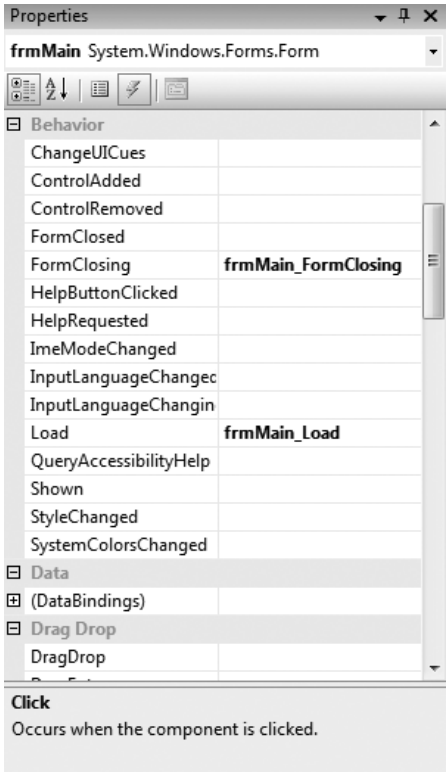

***Рис. 7.6.** Окно свойств frmMain*

```
Private Sub frmMain_Load(ByVal sender As Object, ByVal e As EventArgs) _
  Handles MyBase.Load
  ' запустить WCF-службу
  MyServiceHost.StartService()
End Sub

Private Sub frmMain_FormClosing(ByVal sender As Object, _
                                ByVal e As FormClosingEventArgs) _
  Handles MyBase.FormClosing
  ' остановить WCF-службу
  MyServiceHost.StopService()
End Sub
```

В этом коде есть ссылка на класс `MyServiceHost`. Он запускает WCF-службу и будет подробно рассмотрен ниже.

Платформа Windows Communication Foundation (WCF)

Windows Communication Foundation – это часть платформы .NET Framework, которая позволяет создавать приложения, взаимодействующие между собой по сети. По существу, WCF позволяет вызывать методы объекта, расположенного на другом компьютере, так, будто он работает на локальной машине. В рассматриваемом проекте на одной машине, клиенте, будет работать приложение (в данном случае ASP.NET-приложение), выполняющее метод на другой, хост-машине (host), где работает Outlook и серверное приложение, которое мы собираемся написать.

В проект `WHSMailHost` добавьте ссылку на сборку *System.ServiceModel*. Затем добавьте в проект класс `WHSMailService`. Откройте его и введите приведенный в примерах 7.5 и 7.6 код, поместив его под сгенерированной реализацией класса `WHSMailService`.

Пример 7.5. Класс MyServiceHost (версия на C#)

```
internal class MyServiceHost
{
  internal static ServiceHost myServiceHost = null;

  internal static void StartService()
  {
    // Создать экземпляр ServiceHost
    myServiceHost = new ServiceHost(typeof(WHSMailService));
    myServiceHost.Open();
  }

  internal static void StopService()
  {
    // Метод StopService должен вызываться в процессе
    // освобождения ресурсов (т. е. из метода Dispose)
    if (myServiceHost.State != CommunicationState.Closed)
        myServiceHost.Close();
  }
}
```

Пример 7.6. Класс MyServiceHost (версия на VB)

```
Friend Class MyServiceHost
  Friend Shared myServiceHost As ServiceHost = Nothing

  Friend Shared Sub StartService()
    ' Создать экземпляр ServiceHost
    myServiceHost = New ServiceHost(GetType(WHSMailService))
    myServiceHost.Open()
  End Sub

  Friend Shared Sub StopService()
    ' Метод StopService должен вызываться в процессе
```

```
        ' освобождения ресурсов (т.е. из метода Dispose)
        If myServiceHost.State <> CommunicationState.Closed Then
          myServiceHost.Close()
        End If
      End Sub
    End Class
```

Этот код просто создает экземпляр класса `ServiceHost`, который предоставляет тип `WHSMailService` (его мы реализуем позже), а затем открывает хост, чтобы тот мог принимать входящие соединения. Как эти соединения конфигурируются, мы расскажем ниже.

Напомним, что код `frmMain` вызывал методы `StartService` и `StopService` в обработчиках событий загрузки и закрытия формы соответственно. Таким образом, мы запускаем службу сразу после запуска приложения и останавливаем ее, когда приложение завершается.

Контракты и сущности

В WCF контрактом называется интерфейс, определяющий, какие методы предоставляются службой и могут потребляться клиентским приложением. И клиент, и сервер должны знать, что присутствует в этом интерфейсе, поэтому нам придется создать второй проект, содержащий определение интерфейса.

Кроме того, необходимо предоставить способ передавать информацию о папке и электронном письме между приложениями, поэтому определим классы, инкапсулирующие эти объекты.

Начнем с включения в текущее решение нового проекта типа *Class Library*, который назовем `WHSMailCommon`. Внутри него создайте каталог *Contracts* и каталог *Entities*.

В каталоге *Entities* создайте класс с именем `Folder`. Он будет представлять одну папку Outlook. Его реализация приведена в примерах 7.7 и 7.8.

Пример 7.7. Класс Folder (версия на C#)

```
using System;
using System.Collections.Generic;

namespace WHSMailCommon.Entities
{
  // объект-сущность, представляющий папку
  [Serializable]
  public class Folder : IComparable
  {
    public Folder( string entryID, string name, int unreadMessages,
                   int totalMessages)
    {
      EntryID = entryID;
      Name = name;
```

```
      UnreadMessages = unreadMessages;
      TotalMessages = totalMessages;
    }

    // уникальный идентификатор MAPI
    public string EntryID { get; set; }

    // подпапки этой папки
    public List<Folder> Folders { get; set; }
    public string Name { get; set; }
    public int UnreadMessages { get; set; }
    public int TotalMessages { get; set; }

    // используется для сортировки папок по алфавиту
    public int CompareTo(object obj)
    {
      return string.Compare(this.Name, ((Folder)obj).Name);
    }
  }
}
```

Пример 7.8. Класс Folder (версия на VB)

```
Imports Microsoft.VisualBasic
Imports System
Imports System.Collections.Generic

Namespace WHSMailCommon.Entities
  ' объект-сущность, представляющий папку
  <Serializable> _
  Public Class Folder
    Implements IComparable
    Private _entryID As String
    Private _name As String
    Private _folders As List(Of Folder)
    Private _unreadMessages As Integer
    Private _totalMessages As Integer

    Public Sub New( ByVal entryID As String, ByVal name As String, _
                    ByVal unreadMessages As Integer, _
                    ByVal totalMessages As Integer)
      entryID = entryID
      name = name
      unreadMessages = unreadMessages
      totalMessages = totalMessages
    End Sub

    ' уникальный идентификатор MAPI
    Public Property EntryID() As String
      Get
        Return _entryID
      End Get
      Set(ByVal value As String)
        _entryID = value
```

```
        End Set
      End Property

      ' подпапки этой папки
      Public Property Folders() As List(Of Folder)
        Get
          Return _folders
        End Get
        Set(ByVal value As List(Of Folder))
          _folders = value
        End Set
      End Property

      Public Property Name() As String
        Get
          Return _name
        End Get
        Set(ByVal value As String)
          _name = value
        End Set
      End Property

      Public Property UnreadMessages() As Integer
        Get
          Return Me._unreadMessages
        End Get
        Set(ByVal value As Integer)
          Me._unreadMessages = value
        End Set
      End Property

      Public Property TotalMessages() As Integer
        Get
          Return Me._totalMessages
        End Get
        Set(ByVal value As Integer)
          Me._totalMessages = value
        End Set
      End Property

      ' используется для сортировки папок по алфавиту
      Public Function CompareTo(ByVal obj As Object) As Integer _
          Implements IComparable.CompareTo
        Return String.Compare(Me.Name, (CType(obj, Folder)).Name)
      End Function
    End Class
  End Namespace
```

В этом классе определено несколько свойств, описывающих папку (`EntryID`, `Name` и т. д.), а кроме того, реализован метод `CompareTo` интерфейса `IComparable`, который обеспечивает сортировку папки по алфавиту.

Далее мы создадим класс `Email` (примеры 7.9 и 7.10).

Пример 7.9. Класс Email (версия на C#)

```
using System;
namespace WHSMailCommon.Entities
{
  // объект-сущность, представляющий одно письмо
  [Serializable]
  public class Email
  {
    public Email( string entryID, string from, string fromName,
      string subject, DateTime received, int size)
    {
      EntryID = entryID;
      From = from;
      FromName = fromName;
      Subject = string.IsNullOrEmpty(subject) ? "(no subject)" : subject;
      Received = received;
      Size = size;
    }

    public Email(string entryID, string from, string fromName,
                 string subject, DateTime received, int size, string body) :
                  this(entryID, from, fromName, subject, received, size)
    {
      Body = body;
    }

    // уникальный идентификатор MAPI
    public string EntryID { get; set; }

    // электронный почтовый адрес отправителя
    public string From { get; set; }

    // имя отправителя
    public string FromName { get; set; }

    public string Subject { get; set; }
    public string Body { get; set; }
    public DateTime Received { get; set; }
    public int Size { get; set; }
  }
}
```

Пример 7.10. Класс Email (версия на VB)

```
Imports Microsoft.VisualBasic
Imports System

Namespace WHSMailCommon.Entities
  ' объект-сущность, представляющий одно письмо
  <Serializable> _
  Public Class Email
    Private _entryID As String
    Private _from As String
    Private _fromName As String
```

```
Private _subject As String
Private _received As DateTime
Private _size As Integer
Private _body As String

Public Sub New( ByVal entryID As String, ByVal From As String, _
                ByVal fromName As String, ByVal subject As String, _
                ByVal received As DateTime, ByVal size As Integer)
  entryID = entryID
  from = From
  fromName = fromName
  If String.IsNullOrEmpty(subject) Then
    _subject = "(no subject)"
  Else
    _subject = subject
  End If
  _received = received
  _size = size
End Sub

Public Sub New( ByVal entryID As String, ByVal From As String, _
                ByVal fromName As String, ByVal subject As String, _
                ByVal received As DateTime, ByVal size As Integer, _
                ByVal body As String)
  Me.New(entryID, From, fromName, subject, received, size)
  _body = body
End Sub

' уникальный идентификатор MAPI
Public Property EntryID() As String
  Get
    Return _entryID
  End Get
  Set(ByVal value As String)
    _entryID = value
  End Set
End Property

' электронный почтовый адрес отправителя
Public Property From() As String
  Get
    Return _from
  End Get
  Set(ByVal value As String)
    _from = value
  End Set
End Property

' имя отправителя
Public Property FromName() As String
  Get
    Return _fromName
  End Get
  Set(ByVal value As String)
```

```
        _fromName = value
      End Set
    End Property

    Public Property Subject() As String
      Get
        Return _subject
      End Get
      Set(ByVal value As String)
        _subject = value
      End Set
    End Property

    Public Property Body() As String
      Get
        Return _body
      End Get
      Set(ByVal value As String)
        _body = value
      End Set
    End Property

    Public Property Received() As DateTime
      Get
        Return _received
      End Get
      Set(ByVal value As DateTime)
        _received = value
      End Set
    End Property

    Public Property Size() As Integer
      Get
        Return _size
      End Get
      Set(ByVal value As Integer)
        _size = value
      End Set
    End Property
  End Class
End Namespace
```

Этот класс содержит свойства, описывающие одно электронное письмо: тема (`Subject`), тело (`Body`) и т. д.

Обратите внимание, что оба класса помечены атрибутом `Serializable`. Когда объекты проходят через WCF-службу, они сериализуются отправителем и десериализуются получателем. Если объект снабжен атрибутом `Serializable`, то .NET CLR проделает это автоматически, поскольку в наших объектах-сущностях нет составных типов данных.

Разобравшись с сущностями, мы можем перейти к определению контракта. В каталоге *Contracts* создайте новый файл типа *Interface* с именем *IWHSMailService.cs* (или *.vb*). В этом интерфейсе, описываю-

щем контракт, будут определены три метода: один возвращает дерево объектов `Folder`, представляющее иерархию папок Outlook (`GetFolders`), другой возвращает список объектов `Email` в одной папке (`GetMessages`), а третий – содержимое одного объекта `Email` (`GetMessage`). Код интерфейса представлен в примерах 7.11 и 7.12.

Пример 7.11. Интерфейс IWHSMailService (версия на C#)

```
using System.Collections.Generic;
using System.ServiceModel;
using WHSMailCommon.Entities;

namespace WHSMailCommon.Contracts
{
  // перечень методов службы WHSMailService
  [ServiceContract()]
  public interface IWHSMailService
  {
    [OperationContract]
    List<Folder> GetFolders();

    [OperationContract]
    List<Email> GetMessages(string entryID, int numPerPage, int pageNum);

    [OperationContract]
    Email GetMessage(string entryID);
  }
}
```

Пример 7.12. Интерфейс IWHSMailService (версия на VB)

```
Imports Microsoft.VisualBasic
Imports System.Collections.Generic
Imports System.ServiceModel
Imports WHSMailCommon.Entities

Namespace WHSMailCommon.Contracts
  ' перечень методов службы WHSMailService
  <ServiceContract()> _
  Public Interface IWHSMailService
    <OperationContract> _
    Function GetFolders() As List(Of Folder)

    <OperationContract> _
    Function GetMessages(ByVal entryID As String, _
                         ByVal numPerPage As Integer, _
                         ByVal pageNum As Integer) As List(Of Email)

    <OperationContract> _
    Function GetMessage(ByVal entryID As String) As Email
  End Interface
End Namespace
```

Как и сущности, этот интерфейс также снабжен несколькими атрибутами. Во-первых, любой интерфейс контракта WCF должен быть по-

мечен атрибутом ServiceContract, который говорит WCF, что это именно контракт. Кроме того, все методы, к которым может обращаться клиент (потребляемые методы), должны быть помечены атрибутом OperationContract; иначе их нельзя будет вызвать удаленно.

Код и подробное описание этих методов мы приведем позже, когда дойдем до реализации контракта.

Конфигурирование

Осталось еще сконфигурировать WCF-сервер. Сделать это легко с помощью конфигурационного файла приложения. Добавьте в проект WHSMailHost файл *App.config* типа *Application Configuration*. Заполните его, как показано в примере 7.13.

Пример 7.13. XML-документ в файле app.config

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <system.serviceModel>
    <bindings>
      <netTcpBinding>
        <binding name="NewBinding0">
          <security mode="None" />
        </binding>
      </netTcpBinding>
    </bindings>
    <services>
      <service name="WHSMailHost.WHSMailService">
        <endpoint address="net.tcp://localhost:12345/IWHSMailService"
          binding="netTcpBinding" bindingConfiguration="NewBinding0"
          contract="WHSMailCommon.Contracts.IWHSMailService" />
      </service>
    </services>
  </system.serviceModel>
</configuration>
```

В секции <services> определяются службы, раскрываемые владельцем WCF. Мы создали единственную службу <service> с именем WHSMailHost.WHSMailService; это полное имя класса службы, который мы вскоре реализуем. Внутри тега <service> определена оконечная точка <endpoint>. По существу, оконечная точка – это адрес, с которым соединяется клиент, и его конфигурационные параметры. В данном случае оконечная точка работает по протоколу *net.tcp* и прослушивает порт 12345 на машине localhost в ожидании запросов по контракту IWHSMailService. Далее определяется привязка (binding). Она задает протокол коммуникации со службой, ее конфигурацию и реализуемый контракт. Конфигурационные параметры привязки с именем NewBinding0 (присвоено системой по умолчанию) находятся внутри тега <bindings>. В данном случае задан лишь один параметр: отключить защиту канала между клиентом и сервером. Учитывая, что обе машины будут об-

щаться между собой по локальной сети, о безопасности можно не заботиться. Но если бы речь шла о работе по настоящему Интернет-соединению, которое открыто всему миру, то, конечно, канал следовало бы защитить.

И последний элемент конфигурации службы – это контракт. В нем мы задаем полное имя созданного выше интерфейса `IWHSMailService`.

В этом файле можно сконфигурировать еще мириады разных опций, протоколов, привязок и т. д. и т. п. В конце главы мы приведем несколько ссылок на документацию по платформе WCF, из которой вы можете почерпнуть дополнительную информацию о ее внутреннем устройстве. Отметим также, что для конфигурирования WCF существует графическая программа *Service Configuration Editor*, которая устанавливается в составе Windows SDK или полной версии Visual Studio 2008.

Outlook и MAPI

Интерфейс прикладного программирования электронной почты (Messaging Application Programming Interface – MAPI) представляет собой набор COM-объектов, позволяющих разрабатывать приложения и надстройки над Microsoft Outlook и другими поддерживающими MAPI почтовыми программами (Exchange, Windows Messaging и т. д.). Мы воспользуемся MAPI для получения из Outlook необходимых нам данных.

Выше мы создали контракт `IWHSMailService`. Он будет реализован классом `WHSMailService`. Прежде чем приступать к реализации, вернитесь в проект `WHSMailHost` и установите ссылку на сборку `WHSMailCommon`, входящую в текущее решение. А затем добавьте ссылку на библиотеку типов *Microsoft Outlook 12.0 Object Library*, которую найдете на вкладке `COM` (при условии, что Outlook установлен, как было описано выше) (рис. 7.7).

Далее включите в класс `WHSMailService` пространства имен `Contracts` и `Entities` и сделайте заготовку для реализации интерфейса, как показано в примерах 7.14 и 7.15.

Пример 7.14. Заготовка класса WHSMailService (версия на C#)

```
using WHSMailCommon.Contracts;
using WHSMailCommon.Entities;
using Outlook = Microsoft.Office.Interop.Outlook;

namespace WHSMailHost
{
  public class WHSMailService : IWHSMailService
  {
  }
}
```

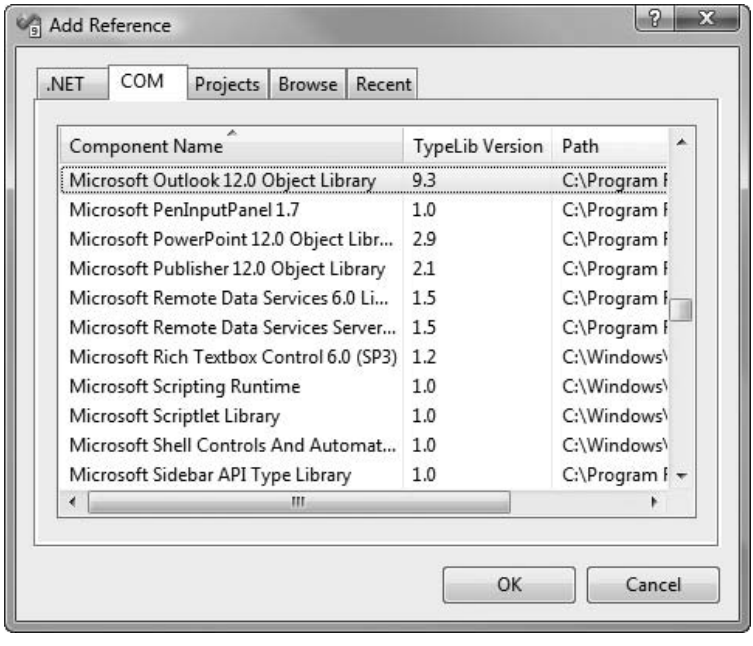

Рис. 7.7. *Ссылка на библиотеку типов Microsoft Outlook 12.0 Object Library*

Пример 7.15. Заготовка класса WHSMailService (версия на VB)

```
Imports WHSMailCommon.Contracts
Imports WHSMailCommon.Entities
Imports Outlook = Microsoft.Office.Interop.Outlook

Public Class WHSMailService
  Implements IWHSMailService
End Class
```

Visual Studio автоматически создает три метода интерфейса, которые следует реализовать: `GetFolders`, `GetMessages` и `GetMessage`. Скоро мы наполним их содержанием. Но сначала напишем конструктор класса. При каждом обращении к службе WCF создает новый экземпляр представляющего ее класса. Поэтому весь код инициализации можно поместить в конструктор без параметров. В данном случае мы инициализируем слой MAPI и выполняем процедуру входа.

Нам необходим экземпляр класса `ApplicationClass`, представляющего Outlook. От него мы можем получить экземпляр пространства имен MAPI. Все методы, которыми мы будем далее пользоваться, принадлежат этому пространству имен. Код конструктора приведен в примерах 7.16 и 7.17.

Пример 7.16. Конструктор класса WHSMailService (версия на C#)

```
private readonly Outlook.NameSpace _nameSpace;

public WHSMailService()
{
  // получить экземпляр пространства имен MAPI и войти в систему
  Outlook.Application app = new Outlook.ApplicationClass();
  _nameSpace = app.GetNamespace("MAPI");
  _nameSpace.Logon(null, null, false, false);
}
```

Пример 7.17. Конструктор класса WHSMailService (версия на VB)

```
Private ReadOnly _nameSpace As Outlook.NameSpace

Public Sub New()
  ' получить экземпляр пространства имен MAPI и войти в систему
  Dim app As Outlook.Application = New Outlook.ApplicationClass()
  _nameSpace = app.GetNamespace("MAPI")
  _nameSpace.Logon(Nothing, Nothing, False, False)
End Sub
```

Имея ссылку на пространство имен, мы можем написать наш метод `GetFolders`. Он получит папку по умолчанию *Inbox* в стандартном хранилище сообщений Outlook, перейдет от нее к родительскому узлу, а потом выполнит рекурсивный обход потомков, отбирая только папки, которые содержат почту, и, наконец, отсортирует их по алфавиту (напомним, что мы раньше реализовали метод `CompareTo` интерфейса `IComparable`). Полный код метода `GetFolders` показан в примерах 7.18 и 7.19.

Пример 7.18. Метод GetFolders (версия на C#)

```
public List<Folder> GetFolders()
{
  List<Folder> list = new List<Folder>();

  // получить папку Inbox и подняться вверх на один уровень...
  // мы *должны* попасть в корень стандартного хранилища
  Outlook.MAPIFolder root = (
             Outlook.MAPIFolder)_nameSpace.GetDefaultFolder(
             Outlook.OlDefaultFolders.olFolderInbox).Parent;

  // добавить корневую папку
  Folder folder = new Folder(root.EntryID, root.Name, root.UnReadItemCount,
                             root.Items.Count);
  list.Add(folder);

  // Обойти подпапки
  EnumerateFolders(root.Folders, folder);

  return list;
}

private void EnumerateFolders(Outlook.Folders folders, Folder rootFolder)
```

```
{
  foreach(Outlook.MAPIFolder f in folders)
  {
    // проверить, что эта папка содержит почтовые сообщения
    // (а не контакты,  условленные встречи и т. д.)
    if(f.DefaultItemType == Outlook.OlItemType.olMailItem)
    {
      if(rootFolder.Folders == null)
         rootFolder.Folders = new List<Folder>();

      // добавить текущую папку и обойти подпапки
      Folder subFolder = new Folder(f.EntryID, f.Name, f.UnReadItemCount,
                                    f.Items.Count);
      rootFolder.Folders.Add(subFolder);
      if(f.Folders.Count > 0)
        this.EnumerateFolders(f.Folders, subFolder);
    }
  }

  // отсортировать список по алфавиту(Folder реализует IComparable)
  rootFolder.Folders.Sort();
}
```

Пример 7.19. Метод GetFolders (версия на VB)

```
Public Function GetFolders() As List(Of Folder) Implements
IWHSMailService.GetFolders
  Dim list As List(Of Folder) = New List(Of Folder)()

  ' получить папку Inbox и подняться вверх на один уровень...
  ' мы *должны* попасть в корень стандартного хранилища
  Dim root As Outlook.MAPIFolder = _
      CType(_nameSpace.GetDefaultFolder( _
           Outlook.OlDefaultFolders.olFolderInbox).Parent, Outlook.MAPIFolder)

  ' добавить корневую папку
  Dim folder As Folder = New Folder( root.EntryID, root.Name, _
                    root.UnReadItemCount, root.Items.Count)
  list.Add(folder)

  ' Обойти подпапки
  EnumerateFolders(root.Folders, folder)

  Return list
End Function

Private Sub EnumerateFolders( ByVal folders As Outlook.Folders, _
                              ByVal rootFolder As Folder)
  For Each f As Outlook.MAPIFolder In folders
    ' проверить, что эта папка содержит почтовые сообщения
    ' (а не контакты,  условленные встречи и т. д.)
    If f.DefaultItemType = Outlook.OlItemType.olMailItem Then
      If rootFolder.Folders Is Nothing Then
        rootFolder.Folders = New List(Of Folder)()
      End If
```

```
        ' добавить текущую папку и обойти подпапки
        Dim subFolder As Folder = New Folder( f.EntryID, f.Name, _
                        f.UnReadItemCount, f.Items.Count)
        rootFolder.Folders.Add(subFolder)
        If f.Folders.Count > 0 Then
          Me.EnumerateFolders(f.Folders, subFolder)
        End If
      End If
    Next f

    ' отсортировать список по алфавиту(Folder реализует IComparable)
    rootFolder.Folders.Sort()
  End Sub
```

Этот метод возвращает универсальный иерархически организованный список `List` объектов `Folder`, который мы отобразим в веб-приложении. Отметим, что одним из свойств объекта `Folder` является `EntryID` – свойство, взятое из объекта `MAPIFolder`. У всех элементов MAPI, будь то почтовые сообщения, папки, условленные встречи и т. д., есть уникальный идентификатор, хранящийся в поле `EntryID`. Он потребуется нам позже для выборки сообщений из конкретной папки.

Далее реализуем метод `GetMessages`. Он будет возвращать список сообщений (без тел сообщений) из папки, задаваемой с помощью метода `GetFolderFromID`. Сообщения сортируются по дате получения в порядке убывания (то есть последнее сообщение оказывается в начале списка). Кроме того, метод осуществляет разбиение на страницы, чтобы не возвращать все содержимое папки за один присест. Реализация представлена в примерах 7.20 и 7.21.

Пример 7.20. Метод GetMessages (версия на C#)

```
public List<Email> GetMessages(string entryID, int numPerPage, int pageNum)
{
  List<Email> list = new List<Email>();

  Outlook.MAPIFolder f;

  // если ID не задан, открываем inbox
  if(string.IsNullOrEmpty(entryID))
    f = _nameSpace.GetDefaultFolder(Outlook.OlDefaultFolders.olFolderInbox);
  else
    f = _nameSpace.GetFolderFromID(entryID, "");

  // для целей сортировки необходимо запомнить свой собственный
  // экземпляр объекта Items
  Outlook.Items items = f.Items;

  // сортировать по времени получения в порядке убывания
  items.Sort("[ReceivedTime]", true);

  // отобрать нужные элементы, исходя из количества элементов
  // на странице и номера текущей страницы
```

```
    for(int i = (numPerPage*pageNum)+1; i <= (numPerPage*pageNum)+numPerPage &&
                                        i <= items.Count; i++)
    {
      // убедиться, что это почтовое сообщение
      Outlook.MailItem mi = (items[i] as Outlook.MailItem);
      if(mi != null)
        list.Add(new Email( mi.EntryID, mi.SenderEmailAddress, mi.SenderName,
                            mi.Subject, mi.ReceivedTime, mi.Size));
    }

    return list;
  }
```

Пример 7.21. Метод GetMessages (версия на VB)

```
  Public Function GetMessages(ByVal entryID As String, _
                              ByVal numPerPage As Integer, _
                              ByVal pageNum As Integer) As List(Of Email) _
                              Implements IWHSMailService.GetMessages
    Dim list As List(Of Email) = New List(Of Email)()

    Dim f As Outlook.MAPIFolder

    ' если ID не задан, открываем inbox
    If String.IsNullOrEmpty(entryID) Then
      f = _nameSpace.GetDefaultFolder(Outlook.OlDefaultFolders.olFolderInbox)
    Else
      f = _nameSpace.GetFolderFromID(entryID, "")
    End If

    ' для целей сортировки необходимо запомнить свой собственный
    ' экземпляр объекта Items
    Dim items As Outlook.Items = f.Items

    ' сортировать по времени получения в порядке убывания
    items.Sort("[ReceivedTime]", True)

    ' отобрать нужные элементы, исходя из количества элементов
    ' на странице и номера текущей страницы
    Dim i As Integer = (numPerPage*pageNum)+1
    Do While i <= (numPerPage*pageNum)+numPerPage AndAlso i <= items.Count
      ' убедиться, что это почтовое сообщение
      Dim mi As Outlook.MailItem = (TryCast(items(i), Outlook.MailItem))
      If Not mi Is Nothing Then
        list.Add(New Email(mi.EntryID, mi.SenderEmailAddress, _
                           mi.SenderName, mi.Subject, mi.ReceivedTime, mi.Size))
      End If
      i += 1
    Loop

    Return list
  End Function
```

Этот метод принимает строковый параметр `entryID` **– уникальный идентификатор папки, из которой мы читаем сообщения. Если идентифи-**

катор не задан, то берутся сообщения из папки *Inbox*. Для разбиения на страницы мы начинаем выборку из массива, возвращаемого свойством `Items` объекта `MAPIFolder`, с указанной позиции и отсчитываем `numPerPage` записей. Метод возвращает список объектов `Email`, которые предстоит отобразить в веб-интерфейсе.

Наконец, реализуем метод `GetMessage`. Он будет возвращать одно сообщение по MAPI-идентификатору `EntryID` с помощью метода `GetItemFromID`. Тело сообщения форматируется в виде обычного текста или HTML, как показано в примерах 7.22 и 7.23.

Пример 7.22. Метод GetMessage (версия на C#)

```
public Email GetMessage(string entryID)
{
  // получить сообщение
  Outlook.MailItem mi =
         (_nameSpace.GetItemFromID(entryID, "") as Outlook.MailItem);

  if (mi != null)
  {
    string body;
    // если это текстовое сообщение, заключим его в тег <pre>
    // для удобства восприятия
    if(mi.BodyFormat == Outlook.OlBodyFormat.olFormatPlain)
      body = "<pre>" + mi.Body + "</pre>";
    else
      body = mi.HTMLBody;

    return new Email( mi.EntryID, mi.SenderEmailAddress, mi.SenderName,
                      mi.Subject, mi.ReceivedTime, mi.Size, body);
  }
  else
    return null;
}
```

Пример 7.23. Метод GetMessage (версия на VB)

```
Public Function GetMessage(ByVal entryID As String) As Email _
        Implements IWHSMailService.GetMessage
    ' получить сообщение
    Dim mi As Outlook.MailItem = _
        (TryCast(_nameSpace.GetItemFromID(entryID, ""), Outlook.MailItem))

    If Not mi Is Nothing Then
        Dim body As String

        ' если это текстовое сообщение, заключим его в тег <pre>
        ' для удобства восприятия
        If mi.BodyFormat = Outlook.OlBodyFormat.olFormatPlain
        Then
            body = "<pre>" & mi.Body & "</pre>"
        Else
            body = mi.HTMLBody
        End If
```

```
            Return New Email(mi.EntryID, mi.SenderEmailAddress, mi.SenderName, _
                             mi.Subject, mi.ReceivedTime, mi.Size, body)
        Else
            Return Nothing
        End If
    End Function
```

Имея эти три метода, мы можем получать список папок, список сообщений в одной папке и одно конкретное сообщение.

На этом разработка хост-приложения завершена. Мы написали приложение Windows Forms, размещенное в области уведомлений, которое получает из Outlook папки и сообщения средствами MAPI, сериализует их и отправляет клиенту для отображения.

Теперь обратимся к веб-приложению, играющему роль клиента, которое мы интегрируем в WHS.

«Клиентское» ASP.NET-приложение

Визуальный интерфейс клиента будет очень похож на Outlook. Имеются три панели: слева – список папок, справа вверху – список сообщений, а справа внизу – текст выбранного сообщения (рис. 7.8).

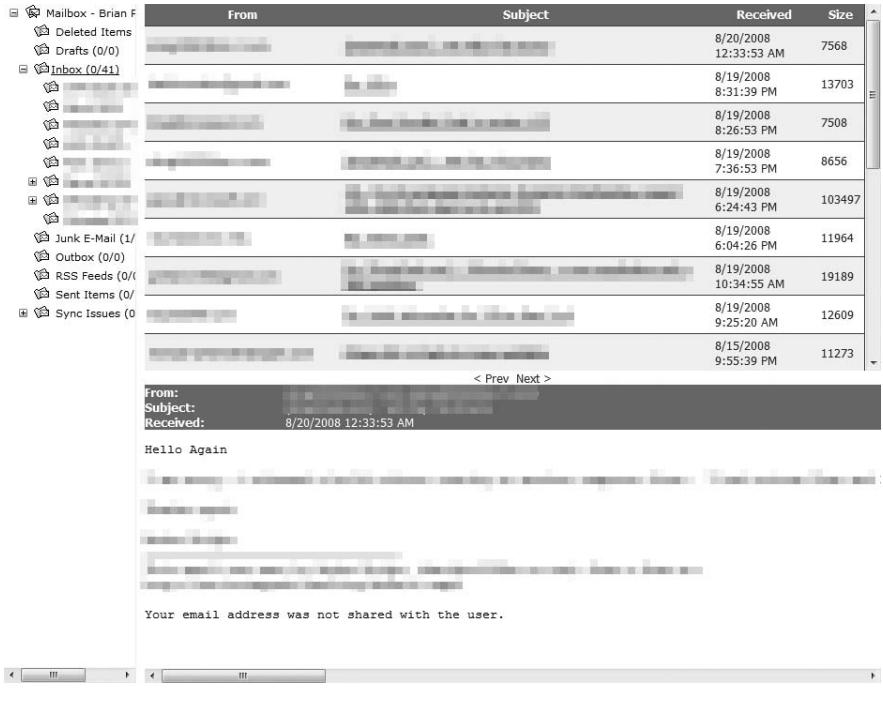

Рис. 7.8. *Интерфейс веб-почты*

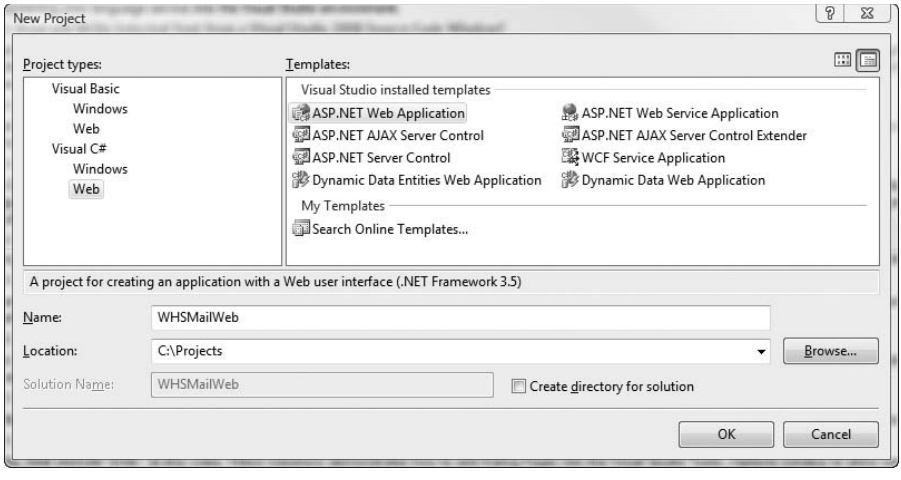

Рис. 7.9. *Создание веб-приложения ASP.NET*

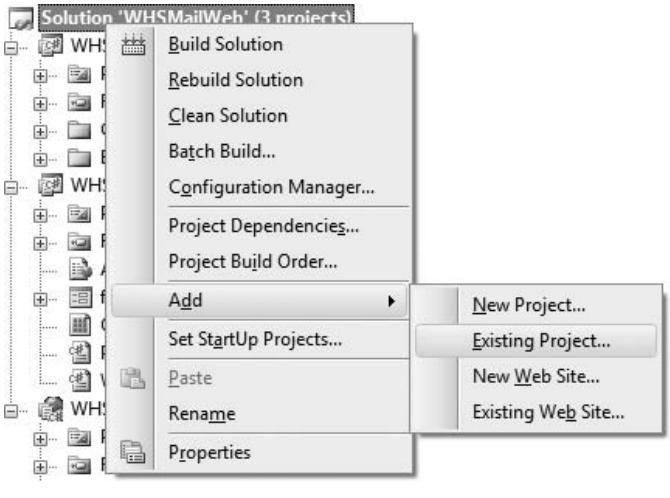

Рис. 7.10. *Добавление существующих проектов*

Откройте Visual Web Developer Express и создайте приложение `WHSMailWeb` типа *ASP.NET Web Application*, как показано на рис. 7.9.

После того как проект будет создан, мы сможем включить в одно с ним решение написанные ранее проекты `WHSMailCommon` и `WHSMailHost`, чтобы все было в одном месте. Проверьте, что оба проекта закрыты в других экземплярах IDE, затем щелкните правой кнопкой мыши по решению `WHSMailWeb`. Из контекстного меню выберите команду Add→Existing

Project... и перейдите в каталог, где находятся проекты WHSMailCommon и WHSMailHost, как показано на рис. 7.10. Эта команда имеется также в меню File в Visual Studio.

Выберите файлы типа *.csproj* или *.vbproj*. Теперь проекты появятся на панели Solution Explorer, и вы сможете редактировать и собирать их, не покидая данного решения.

Вернитесь в проект WHSMailWeb и добавьте ссылку на проект WHSMailCommon, а также на сборку System.ServiceModel.

Теперь сконфигурируем ASP.NET-приложение, так чтобы оно правильно вызывало WCF-службу, работающую на сервере. Откройте файл *web.config* и замените его содержимое кодом из примера 7.24.

Пример 7.24. XML-документ в файле web.config

```
<?xml version="1.0"?>
<configuration>
    <system.web>
        <compilation debug="true" />
        <customErrors mode="Off" />
        <!-- ВСТАВЬТЕ СЮДА КОД ИЗ /REMOTE/WEB.CONFIG!!-->
        <!-- НЕ ЗАБУДЬТЕ ИЗМЕНИТЬ ЭЛЕМЕНТЫ FORMS И CUSTOMERRORS! -->

        <!-- КОНЕЦ СКОПИРОВАННОГО КОДА-->
    </system.web>
    <system.serviceModel>
        <bindings>
            <netTcpBinding>
                <binding name="NewBinding0" maxReceivedMessageSize="1048576">
                    <readerQuotas maxStringContentLength="1048576" />
                    <security mode="None" />
                </binding>
            </netTcpBinding>
        </bindings>
        <client>
            <endpoint address="net.tcp://SERVERNAME:12345/IWHSMailService"
                binding="netTcpBinding" bindingConfiguration="NewBinding0"
                contract="WHSMailCommon.Contracts.IWHSMailService"
                name="WHSMailService" />
        </client>
    </system.serviceModel>
</configuration>
```

Все это очень похоже на конфигурирование WCF-службы. Обратите внимание, что оконечная точка указывает на машину, где эта служба работает.

Примечание

Вместо строки SERVERNAME необходимо подставить имя или IP-адрес машины, на которой работает Outlook и приложение WHSMailHost.

Привязка и контракт сконфигурированы точно так же, как для самой службы. Ниже мы воспользуемся параметром `name` для того, чтобы WCF знал, как обращаться к владельцу службы для создания удаленного объекта.

Отличается только конфигурация элемента `netTcpBinding`. Режим безопасности, как и раньше, равен `none`, но добавились атрибуты `maxReceivedMessageSize` и `maxStringContentLength`. Оба равны 1 мегабайту – это максимальный размер сообщения, передаваемого от сервера клиенту. По умолчанию эта величина составляет всего 64 Кбайт, что явно недостаточно для тех объемов данных, которые нам предстоит сериализовать и передать.

Теперь добавьте в проект класс `BasePage`. Этой странице будут наследовать все остальные страницы в проекте, чтобы упростить открытие и закрытие WCF-канала, необходимого для получения данных о почте. Реализуйте класс `BasePage`, как показано в примерах 7.25 и 7.26.

Пример 7.25. Страница BasePage (версия на C#)

```
using System;
using System.ServiceModel;
using WHSMailCommon.Contracts;

namespace WHSMailWeb
{
  public partial class BasePage : System.Web.UI.Page
  {
    ChannelFactory<IWHSMailService> _factory = null;
    private IWHSMailService _channel = null;

    protected void Page_Init(object sender, EventArgs e)
    {
      // создать фабрику каналов и получить от нее
      // прокси-канал
     _factory = new ChannelFactory<IWHSMailService>("WHSMailService");
     _channel = _factory.CreateChannel();
    }

    protected void Page_Unload(object sender, EventArgs e)
    {
      try
      {
       _factory.Close();
      }
      catch
      {
        // пока нас не волнует, что здесь происходит...
        // если ошибка, то так тому и быть
      }
    }

    public IWHSMailService WHSMailService
```

```
    {
      get { return _channel; }
    }
  }
}
```

Пример 7.26. Страница BasePage (версия на VB)

```
Imports Microsoft.VisualBasic
Imports System
Imports System.ServiceModel
Imports WHSMailCommon.Contracts

Namespace WHSMailWeb
  Public Partial Class BasePage
        Inherits System.Web.UI.Page
    Private _factory As ChannelFactory(Of IWHSMailService) = Nothing
    Private _channel As IWHSMailService = Nothing

    Protected Sub Page_Init(ByVal sender As Object, ByVal e As EventArgs)
      ' создать фабрику каналов и получить от нее прокси-канал
      _factory = New ChannelFactory(Of IWHSMailService)("WHSMailService")
      _channel = _factory.CreateChannel()
    End Sub

    Protected Sub Page_Unload(ByVal sender As Object, ByVal e As EventArgs)
      Try
        _factory.Close()
      Catch
        ' пока нас не волнует, что здесь происходит...
        ' если ошибка, то так тому и быть
      End Try
    End Sub

    Public ReadOnly Property WHSMailService() As IWHSMailService
      Get
        Return _channel
      End Get
    End Property

  End Class
End Namespace
```

Здесь мы переопределяем методы страницы `Init` и `Unload`. `Page_Init` вызывается в начале обработки запроса к странице, а `Page_Unload` — непосредственно перед ее завершением.

В данном случае метод `Page_Init` использует WCF, чтобы создать фабричный объект `ChannelFactory`, который умеет порождать прокси-объекты типа `IWHSMailService`. Напомним, что этот интерфейс определяет контракт службы. Имя, передаваемое конструктору `ChannelFactory` в качестве параметра, должно совпадать с именем службы в рассмотренном выше конфигурационном файле.

Затем создается экземпляр канала путем вызова метода `CreateChannel` объекта `ChannelFactory`. Это и есть прокси-объект, оперделенный нашим контрактом. Его возвращает свойство `WHSMailService`, описанное в классе последним, благодаря чему все производные страницы могут без труда получить доступ к объекту канала, чтобы вызвать службу. Как это делается, мы увидим чуть ниже.

Метод `Page_Unload` закрывает фабрику и весьма легкомысленно обрабатывает исключения, которые при этом могут возникнуть. В этом приложении ошибка при закрытии не критична, но так дело обстоит далеко не всегда.

Теперь реализуем страницу *Default.aspx*. Ее HTML-разметка показана в примере 7.27.

Пример 7.27. HTML-разметка страницы Default.aspx

```
<%@ Page Language="C#" AutoEventWireup="true" CodeBehind="Default.aspx.cs"
Inherits="WHSMailWeb._Default" %>
<%@ Import namespace="System.ComponentModel"%>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml" >
<head runat="server">
    <title>WHS Email</title>
    <style>
body
{
    margin:0;
    padding:0;
    height:100%;
}

#top
{
    position: absolute;
    left: 16%;
    top: 0px;
    margin: 0px;
    padding: 0px;
    overflow: auto;
    height:54%;
    width:84%;
}

#pager
{
    position: absolute;
    left: 16%;
    top: 54%;
    margin: 0px;
    padding: 0px;
```

```
    overflow: auto;
    height: 24px;
    width: 84%;
    text-align: center;
    font-family: Tahoma;
    font-size: Smaller;
}

#bottom
{
    position: absolute;
    left: 16%;
    margin: 0px;
    padding: 0px;
    vertical-align: top;
    top: 56%;
    height: 44%;
    width: 84%;
    overflow: auto;
    border-top: thin solid #EFF3FB;
    word-wrap:break-word;
}

#folders
{
    position: absolute;
    left: 0px;
    margin: 0px;
    padding: 0px;
    vertical-align: top;
    height: 100%;
    width: 15%;
    overflow: auto;
    border-right: thin solid #EFF3FB;
}

#tblHeader
{
    color:#FFFFFF;
    font-family: Tahoma;
    font-size: smaller;
}
    </style>
</head>
<body scroll="no">
    <form id="form1" runat="server">
    <div id="folders">
      <asp :TreeView ID="tvFolders" runat="server"
       OnSelectedNodeChanged="tvFolders_SelectedNodeChanged" ExpandDepth="0"
       ImageSet="Inbox" NodeIndent="10">
       <ParentNodeStyle Font-Bold="False" />
       <HoverNodeStyle Font-Underline="True" />
      <SelectedNodeStyle Font-Underline="True" HorizontalPadding="0px"
```

```
      VerticalPadding="0px" />
    <NodeStyle Font-Names="Verdana" Font-Size="8pt" ForeColor="Black"
      HorizontalPadding="5px"
     NodeSpacing="0px" VerticalPadding="0px" />
    </asp:TreeView>
  </div>
  <div id="top">
    <asp :GridView ID="GridView1" runat="server" CellPadding="4"
      ForeColor="#333333" GridLines="Horizontal" AutoGenerateColumns="False"
      CaptionAlign="Top" HorizontalAlign="Center" Width="100%" PageSize="20"
          Font-Names="Tahoma" Font-Size="Small">
      <FooterStyle BackColor="#507CD1" Font-Bold="True" ForeColor="White" />
      <RowStyle BackColor="#EFF3FB" />
      <EditRowStyle BackColor="#2461BF" />
      <SelectedRowStyle BackColor="#D1DDF1" Font-Bold="True"
          ForeColor="#333333" />
      <PagerStyle BackColor="#2461BF" ForeColor="White"
          HorizontalAlign="Center" />
      <HeaderStyle BackColor="#507CD1" Font-Bold="True" ForeColor="White" />
      <AlternatingRowStyle BackColor="White" />
      <Columns>
        <asp:BoundField DataField="From" HeaderText="From" />
        <asp:TemplateField HeaderText="Subject">
          <ItemTemplate>
            <asp :LinkButton ID="btnLink" OnCommand="btnLink_Command"
                CommandArgument='<%#Eval("EntryID")%>'
                runat="server"><%#Eval("Subject")%></asp:LinkButton>
          </ItemTemplate>
        </asp:TemplateField>
        <asp:BoundField DataField="Received" HeaderText="Received" />
        <asp:BoundField DataField="Size" HeaderText="Size" />
      </Columns>
    </asp:GridView>
  </div>
  <div id="pager">
    <asp :LinkButton ID="btnPrev" runat="server"
        OnClick="btnPrev_Click">&lt; Prev</asp:LinkButton> 
    <asp :LinkButton ID="btnNext" runat="server"
        OnClick="btnNext_Click">Next &gt;</asp:LinkButton>
  </div>
  <div id="bottom" runat="server">
    <table cellpadding="0" cellspacing="0" id="tblHeader" bgcolor="#507CD1"
        width="100%" runat="server">
      <tr><td><strong>From:</strong></td><td><asp:Label ID="lblFrom"
          runat="server"/></td></tr>
       <tr><td><strong>Subject:</strong></td><td><asp:Label ID="lblSubject"
          runat="server"/></td></tr>
      <tr><td><strong>Received:</strong></td><td><asp:Label ID="
          lblReceived" runat="server"/></td></tr>
    </table>
    <div id="msgContent" runat="server"></div>
```

```
    </div>
  </form>
</body>
</html>
```

На этой странице есть четыре тега <DIV>: один содержит объект ASP.NET TreeView (левая панель), второй – объект ASP.NET GridView (правая верхняя панель), третий – два объекта ASP.NET LinkButton, реализующие листание страниц вперед и назад (правая средняя панель), а последний – таблицу, содержащую заголовки и тег <DIV>, в котором отображается сообщение (правая нижняя панель).

Для элемента GridView задан набор стилей, а вслед за ними информация о колонках в теге <Columns>. Сетка состоит из четырех колонок: адрес отправителя (From), тема (Subject), дата получения сообщения (Received) и его размер (Size). Колонки From, Received и Size имеют тип BoundField. Они автоматически создаются в результате привязки к данным, о которой мы поговорим чуть ниже. Колонка Subject имеет тип TemplateField, это позволяет включить в каждую строку элемент LinkButton, так что тема становится гиперссылкой, при щелчке по которой в нижней панели отображается текст сообщения.

Как мы уже видели, в результате получается веб-страница, изображенная на рис. 7.11.

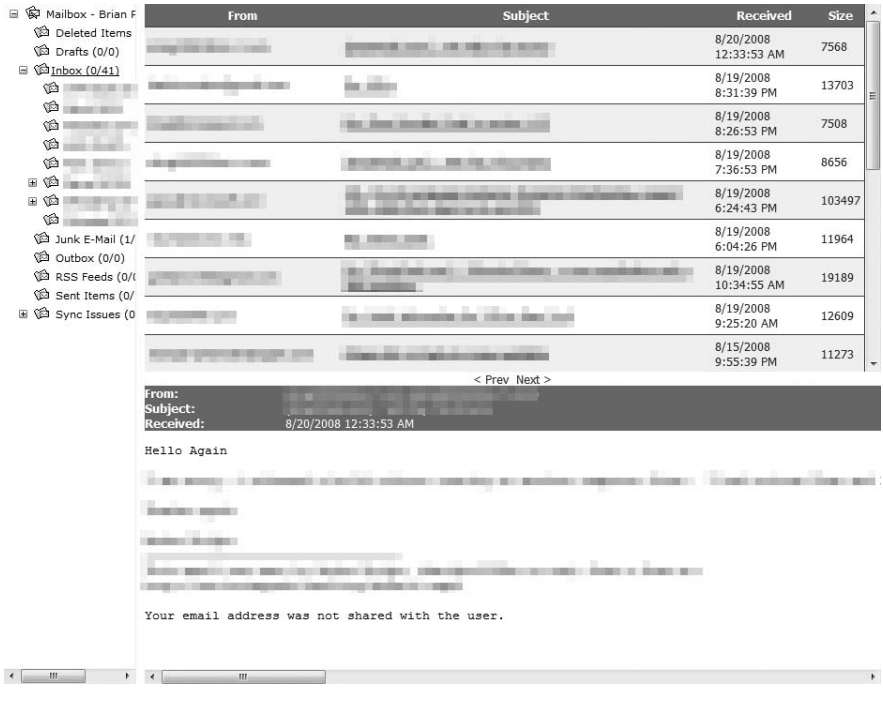

Рис. 7.11. *Интерфейс веб-почты*

Теперь самое время реализовать логику работы страницы. Откройте код программной логики страницы *Default.aspx* и введите заготовку, показанную в примерах 7.28 и 7.29.

Пример 7.28. Код программной логики, класс _Default (версия на C#)

```
using System;
using System.Collections.Generic;
using System.Web.UI.WebControls;
using WHSMailCommon.Entities;

namespace WHSMailWeb
{
  public partial class _Default : BasePage
  {
  }
}
```

Пример 7.29. Код программной логики, класс _Default (версия на VB)

```
Imports Microsoft.VisualBasic
Imports System
Imports System.Collections.Generic
Imports System.Web.UI.WebControls
Imports WHSMailCommon.Entities

Namespace WHSMailWeb
  Public Partial Class _Default
      Inherits BasePage
  End Class
End Namespace
```

Теперь реализуем метод `Page_Load`. Он вызывается после метода `Page_Init`, который мы реализовали в родительском классе. Код показан в примерах 7.30 и 7.31

Пример 7.30. Метод Page_Load и вспомогательные методы (версия на C#)

```
private string _folderEntryID = string.Empty;
private int _pageNum = 0;

protected void Page_Load(object sender, EventArgs e)
{
  // при первом вызове (для tvFolders включено Viewstate)
  if(tvFolders.Nodes.Count == 0)
  {
    // получить дерево папок
    List<Folder> folderList = this.WHSMailService.GetFolders();

    // добавить корень и раскрыть его
    TreeNode node = new TreeNode(folderList[0].Name, folderList[0].EntryID);
    node.Expanded = true;
    tvFolders.Nodes.Add(node);

    // загрузить подпапки
```

```
    LoadNode(folderList[0].Folders, node);

    // получить список сообщений в inbox и привязать его к сетке
    List<Email> list = GetMessages();
    BindMessages(list);

    // запомнить номер страницы и идентификатор папки
    ViewState["PageNum"] = _pageNum;
    ViewState["FolderID"] = _folderEntryID;
  }

  _pageNum = int.Parse(ViewState["PageNum"].ToString());
  _folderEntryID = ViewState["FolderID"].ToString();
}

private void LoadNode(List<Folder> folders, TreeNode node)
{
  foreach(Folder f in folders)
  {
    // добавить форматированный узел: имя папки + (Не прочитано/всего)
    TreeNode subNode = new TreeNode(f.Name + " (" + f.UnreadMessages + "/" +
                                    f.TotalMessages + ")", f.EntryID);

    // раскрыть и выбрать папку inbox
    subNode.Expanded = subNode.Selected = (f.Name == "Inbox");
    node.ChildNodes.Add(subNode);

    // загрузить подпапки
    if(f.Folders != null)
      LoadNode(f.Folders, subNode);
  }
}

private List<Email> GetMessages()
{
  // получить группу сообщений, зная номер текущей страницы
  // и размер страницы
  return this.WHSMailService.GetMessages(_folderEntryID, GridView1.PageSize,
                                         _pageNum);
}

private void BindMessages(List<Email> list)
{
  // загрузить сетку
  GridView1.DataSource = list;
  GridView1.DataBind();

  // запомнить номер новой страницы
  ViewState["PageNum"] = _pageNum;
}
```

Пример 7.31. Метод Page_Load и вспомогательные методы (версия на VB)

```
Private _folderEntryID As String = String.Empty
Private _pageNum As Integer = 0
```

```
Protected Sub Page_Load(ByVal sender As Object, ByVal e As EventArgs)
    ' при первом вызове (для tvFolders включено Viewstate)
    If tvFolders.Nodes.Count = 0 Then
    ' получить дерево папок
    Dim folderList As List(Of Folder) = Me.WHSMailService.GetFolders()

    ' добавить корень и раскрыть его
    Dim node As TreeNode = New TreeNode(folderList(0).Name, _
                                        folderList(0).EntryID)
    node.Expanded = True
    tvFolders.Nodes.Add(node)

    ' загрузить подпапки
    LoadNode(folderList(0).Folders, node)

    ' получить список сообщений в inbox и привязать его к сетке
    Dim list As List(Of Email) = GetMessages()
    BindMessages(list)

    ' запомнить номер страницы и идентификатор папки
    ViewState("PageNum") = _pageNum
    ViewState("FolderID") = _folderEntryID
  End If

  _pageNum = Integer.Parse(ViewState("PageNum").ToString())
  _folderEntryID = ViewState("FolderID").ToString()
End Sub

Private Sub LoadNode(ByVal folders As List(Of Folder), _
                     ByVal node As TreeNode)
  For Each f As Folder In folders
    ' добавить форматированный узел: имя папки + (Не прочитано/всего)
    Dim subNode As TreeNode = New TreeNode(f.Name & " (" & f.UnreadMessages & _
                                        "/" & f.TotalMessages & ")", _
                                        f.EntryID)
    ' раскрыть и выбрать папку inbox
    subNode.Selected = (f.Name = "Inbox")
    subNode.Expanded = subNode.Selected
    node.ChildNodes.Add(subNode)

    ' загрузить подпапки
    If Not f.Folders Is Nothing Then
            LoadNode(f.Folders, subNode)
    End If
  Next f
End Sub

Private Function GetMessages() As List(Of Email)
  ' получить группу сообщений, зная номер текущей страницы
  ' и размер страницы
  Return Me.WHSMailService.GetMessages( folderEntryID, GridView1.PageSize,
                    _pageNum)
End Function

Private Sub BindMessages(ByVal list As List(Of Email))
```

```
    ' загрузить сетку
    GridView1.DataSource = list
    GridView1.DataBind()

    ' запомнить номер новой страницы
    ViewState("PageNum") = _pageNum
  End Sub
```

Если дерево tvFolders еще не заполнено, то мы вызываем метод GetFolders нашей WCF-службы, пользуясь созданным ранее каналом, который хранится в свойстве WHSMailService. Затем этот метод обходит возвращенный иерархический список папок, строя древовидную структуру, соответствующую имеющейся в Outlook. Свойству Text каждого узла TreeNode присваивается строка, состоящая из имени папки и двух чисел: количества непрочитанных сообщений в папке и общего количества сообщений в ней. В свойство узла Value записывается уникальный идентификатор MAPI EntryID, чтобы позже мы по этому значению могли получить список сообщений в данной папке.

Далее мы вызываем метод службы GetMessages, который возвращает сообщения из папки Inbox. Мы передаем значение, хранящееся в свойстве PageSize сетки, и переменную-член _pageNum. Это позволяет реализовать описанную выше схему разбиения на страницы. После того как список сообщений получен, он связывается с сеткой с данными для отображения. Все колонки типа BoundField при этом заполняются автоматически, без каких бы то ни было усилий с нашей стороны.

И наконец, мы сохраняем в контейнере ViewState значения по умолчанию номера страницы (0) и идентификатора папки (""), чтобы при следующем запросе их можно было извлечь и присвоить переменным-членам нашего класса.

Следующим шагом мы реализуем реакцию на щелчок пользователя по папке, представленной в дереве. Нам нужно просто вытащить идентификатор MAPI EntryID (он хранится в поле Value выбранного узла), присвоить его локальной переменной-члену и записать в ViewState, после чего вызвать метод службы GetMessages, который вернет список сообщений в данной папке. Точно так же мы выше получали сообщения из папки Inbox. Соответствующий код представлен в примерах 7.32 и 7.33.

Пример 7.32. Обработчик события tvFolders_SelectedNodeChanged (версия на C#)

```
protected void tvFolders_SelectedNodeChanged(object sender, EventArgs e)
{
  // новая папка, перестроить изображение
  _pageNum = 0;
  _folderEntryID = this.tvFolders.SelectedNode.Value;
  ViewState["FolderID"] = _folderEntryID;
  List<Email> list = GetMessages();
  BindMessages(list);
}
```

Пример 7.33. Обработчик события tvFolders_SelectedNodeChanged (версия на VB)

```
Protected Sub tvFolders_SelectedNodeChanged( ByVal sender As Object, _
                                             ByVal e As EventArgs)
  ' новая папка, перестроить изображение
  _pageNum = 0
  _folderEntryID = Me.tvFolders.SelectedNode.Value
  ViewState("FolderID") = _folderEntryID

  Dim list As List(Of Email) = GetMessages()
  BindMessages(list)
End Sub
```

Теперь можно реализовать обработчики нажатий на кнопки Next и Back (примеры 7.34 и 7.35). Эти методы увеличивают или уменьшают номер текущей страницы, записывают результат в контейнер `ViewState` **для следующего запроса, а затем, как и раньше, получают список сообщений из текущей папки, используя ее идентификатор, хранящийся в** `ViewState`**.**

Пример 7.34. Обработчики событий btnPrev_Click и btnNext_Click (версия на C#)

```
protected void btnPrev_Click(object sender, EventArgs e)
{
  // предыдущая страница
  if(_pageNum > 0)
     _pageNum--;

  List<Email> list = GetMessages();
  BindMessages(list);
}

protected void btnNext_Click(object sender, EventArgs e)
{
  // следующая страница
  _pageNum++;
  List<Email> list = GetMessages();

  // если сообщений больше нет, вернуться к предыдущей странице
  if(list == null || list.Count == 0)
  {
    _pageNum--;
     list = GetMessages();
  }
  BindMessages(list);
}
```

Пример 7.35. Обработчики событий btnPrev_Click и btnNext_Click (версия на VB)

```
Protected Sub btnPrev_Click(ByVal sender As Object, ByVal e As EventArgs)
  ' предыдущая страница
  If _pageNum > 0 Then
```

```
      _pageNum -= 1
  End If

  Dim list As List(Of Email) = GetMessages()
  BindMessages(list)
End Sub

Protected Sub btnNext_Click(ByVal sender As Object, ByVal e As EventArgs)
  ' следующая страница
  _pageNum += 1
  Dim list As List(Of Email) = GetMessages()

  ' если сообщений больше нет, вернуться к предыдущей странице
  If list Is Nothing OrElse list.Count = 0 Then
    _pageNum -= 1
    list = GetMessages()
  End If
  BindMessages(list)
End Sub
```

А теперь нужно реализовать вывод тела сообщения, которое пользователь выбрал на верхней панели. Взглянув на HTML-код страницы *Default.aspx*, вы обнаружите, что поле `Subject` привязано к элементу управления `LinkButton` в сетке с помощью шаблонной колонки `TemplateField` (остальные колонки имеют стандартный тип `BoundField`). Для элемента `LinkButton` в свойство `CommandArgument` записан уникальный MAPI-идентификатор `EntryID` данного сообщения. Поэтому нам остается лишь обработать событие `Command` от элемента `LinkButton`, извлечь идентификатор из аргумента события и вызвать метод нашей службы `GetMessage`, передав ему этот идентификатор. А уж он вернет само сообщение (см. примеры 7.36 и 7.37).

Пример 7.36. Обработчик события btnLink_Command (версия на C#)

```
protected void btnLink_Command(object sender, CommandEventArgs e)
{
  Email email =
this.WHSMailService.GetMessage(e.CommandArgument.ToString());

  // сформировать заголовки
  lblFrom.Text = email.FromName + " (" + email.From + ")";
  lblSubject.Text = email.Subject;
  lblReceived.Text = email.Received.ToString();

  // вывести текст выбранного сообщения
  msgContent.InnerHtml = email.Body;
}
```

Пример 7.37. Обработчик события btnLink_Command (версия на VB)

```
Protected Sub btnLink_Command(ByVal sender As Object, _
                              ByVal e As CommandEventArgs)
  Dim email As Email = _
  Me.WHSMailService.GetMessage(e.CommandArgument.ToString())
```

```
    ' сформировать заголовки
    lblFrom.Text = email.FromName & " (" & email.From & ")"
    lblSubject.Text = email.Subject
    lblReceived.Text = email.Received.ToString()

    ' вывести текст выбранного сообщения
    msgContent.InnerHtml = email.Body
End Sub
```

Мы получаем объект-сущность `Email`, копируем его свойства в метки, составляющие заголовок, а тело сообщения – в свойство `InnerHtml` соответствующего элемента `<DIV>`.

Voilá! Очень простая программа чтения почты готова.

Итак, код написан. Осталось развернуть приложения и сконфигурировать Windows Home Server.

Внимание

Следует упомянуть об одном подводном камне. Веб-клиент отображает содержимое сообщения без строгих проверок на безопасность, которые сделал бы Outlook. Поэтому будьте осторожны, открывая сообщения неизвестного происхождения и щелкая по неизвестным ссылкам, поскольку Outlook'а, который мог бы вас оберечь, здесь нет.

Развертывание

Для начала установите .NET Framework 3.5 на ту машину, где будет работать приложение `WHSMailHost`, если это еще не сделано. Затем скопируйте файлы *WHSMailHost.exe*, *WHSMailHost.exe.config* и *WHSMailCommon.dll* в какой-нибудь каталог на этой машине. Возможно, вы захотите создать ярлык в группе Пуск, чтобы приложение автоматически запускалось при входе в Windows.

Затем откройте Outlook и выберите из меню пункт Сервис→Центр доверия (Tools→Trust Center). Перейдите в раздел Доступ из программ (Programmatic Access) и отметьте переключатель «Never warn me about suspicious activity» (Никогда не предупреждать о подозрительной активности), как показано на рис. 7.12. К сожалению, эту меру безопасности придется отключить, иначе Outlook будет запрашивать подтверждение при любой попытке владельца службы прочитать данные. Не существует способа разрешить доступ какому-то одному приложению – приходится разрешать всем сразу.

Примечание

Если вы работаете в Windows Vista, то для изменения этого параметра должны будете запустить Outlook от имени администратора. Изменив его, можете дальше запускать Outlook от имени обычного пользователя.

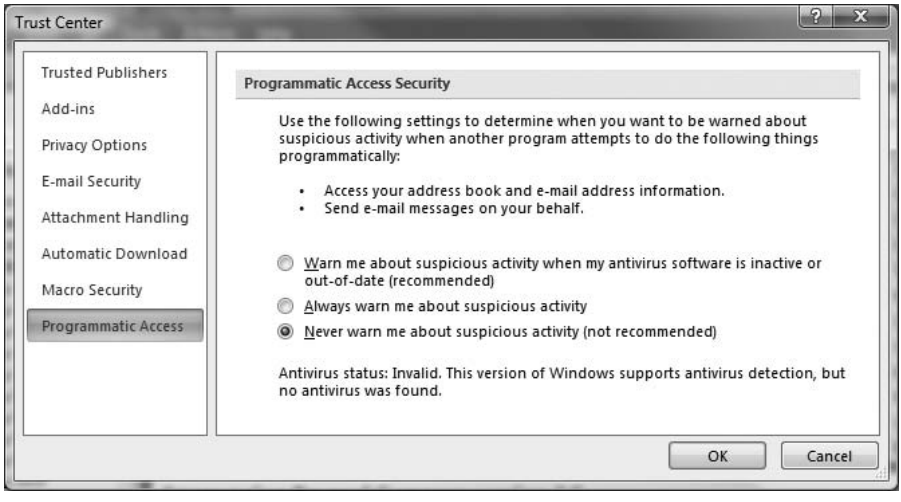

Рис. 7.12. *Настройка безопасного доступа из программ в Outlook 2007*

Если на вашем ПК работает брандмауэр, то откройте порт 12345. Можете изменить номер 12345 на какой-нибудь другой, но не забудьте внести изменение в конфигурационные файлы обоих приложений.

И еще отметим, что для работы этого приложения Outlook не обязательно должен быть постоянно открыт. Однако вы все же должны войти в систему на машине, где работает владелец службы.

WHSMailWeb

Для выполнения последующих шагов вам понадобится установить соединение с удаленным рабочим столом на Windows Home Server, пользуясь программой Remote Desktop, а не Windows Home Server Connector.

На машине под управлением Windows Home Server установите исполняющую среду .NET Framework 3.5 Runtime, без нее WCF доступен не будет. Затем создайте каталог *mail* в уже существующем каталоге *c:\inetpub*. Скопируйте туда все ASPX-страницы, а также файлы *web.config*, *icon.png* и каталог *bin*. Затем откройте приложение Internet Information Services (IIS) Manager, находящееся в группе Администрирование (Administrative Tools) в меню Пуск. Раскройте узел Веб-узлы (Web Sites), щелкните правой кнопкой мыши по узлу Веб-узел по умолчанию (Default Web Site) и выберите из контекстного меню команду Создать→Виртуальный каталог (New→Virtual Directory...), как показано на рис. 7.13.

Заполните предлагаемые поля, как показано в табл. 7.2.

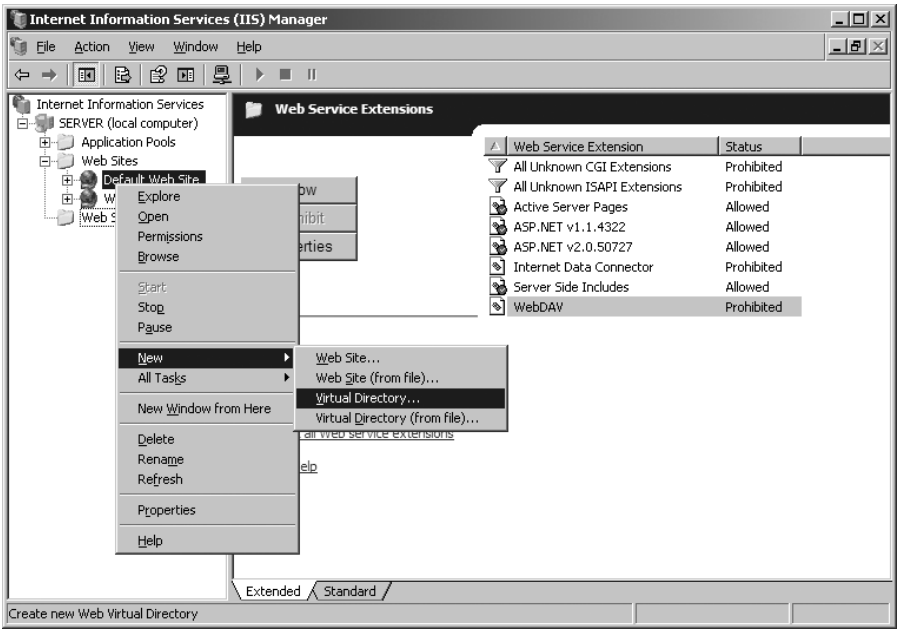

Рис. 7.13. *Создание виртуального каталога в IIS*

Таблица 7.2. Свойства виртуального каталога

Свойство	**Значение**
Псевдоним	Mail
Каталог	C:\inetpub\mail
Разрешить	Чтение, запуск сценариев

Следующий шаг чуть посложнее. Мы хотим воспользоваться интегрированным режимом безопасности, который предоставляет Windows Home Server. *Удаленный* веб-сайт в системе WHS пользуется механизмом безопасности на уровне форм, встроенным в ASP.NET. В идеале хотелось бы заходить на главную страницу удаленного сайта на сервере WHS, а затем открывать ссылку на веб-почту, не регистрируясь еще раз. Если же вы пытаетесь обратиться к URL *почты* напрямую, то должны запрашиваться имя и пароль.

Чтобы реализовать эту идею, откройте в Блокноте файл *web.config* из каталога *c:\inetpub\remote*. В XML-разметке между открывающим и закрывающим тегами `<system.web>` находятся конфигурационные параметры механизма аутентификации на уровне форм, который применяется в WHS. Чтобы воспользоваться cookie, который создает этот механизм аутентификации, нам необходимо иметь те же значения ключей в нашем файле *web.config*. Проще всего добиться этого, скопировав весь текст между открывающим и закрывающим тегами `<system.web>`

из файла *web.config* в каталоге remote. Затем откройте недавно скопированный файл *web.config* в каталоге *mail* и вставьте этот текст между двумя строками с соответствующей подсказкой. В примере 7.38 показано, что должно получиться, но *не используйте именно этот код*, – у вас он не заработает. Значения на каждой машине уникальны.

Пример 7.38. XML-код, копируемый из одного файла web.config в другой

```
<machineKey validationKey="
24032958BBC3088B5B117035A7399814727BA88188A044271034A21968
FFCA1621D8F923102D345F9166A42E4F6B6A70F4B838F8127F844D2AB0C492F43AE2F6"
  decryptionKey="3D8D5A023C289F4E19085134E1CACA0842BC363122B058EFD8E6CF7
  96E4406E3"
  validation="SHA1" decryption="AES" />
  <authentication mode="Forms">
       <forms name="RemotePortalAuth" loginUrl="logon.aspx" protection="All"
        path="/" timeout="12000" requireSSL="false"/>
    </authentication>
    <authorization>
        <deny users="?"/>
        <allow users="*"/>
    </authorization>
    <httpRuntime executionTimeout="86400" maxRequestLength="2097151" />
    <customErrors mode="On" defaultRedirect="error.aspx"/>
    <trace enabled="false" localOnly="false" pageOutput="false"
      requestLimit="100" traceMode="SortByTime" />
    <sessionState mode="InProc" cookieless="false" timeout="20"/>
    <globalization requestEncoding="utf-8" responseEncoding="utf-8"/>
```

Еще необходимо изменить пути `loginUrl` и `defaultRedirect` в элементах `forms` и `customErrors` в файле *web.config*. Добавьте в начало каждого пути строку `/remote/` – получатся соответственно `/remote/login.aspx` и `/remote/error.aspx`, как показано в примере 7.39.

Пример 7.39. Измененные элементы <forms> и <customErrors>

```
<forms name="RemotePortalAuth" loginUrl="/remote/logon.aspx" protection="All"
  path="/" timeout="12000" requireSSL="false"/>
<customErrors mode="On" defaultRedirect="/remote/error.aspx"/>
```

Наконец, не забудьте заменить строку `SERVERNAME` в конфигурации оконечной точки именем или IP-адресом машины, на которой работает процесс-владелец службы.

Скопировав и отредактировав все файлы, убедитесь, что на них распространяются разрешения, заданные для объемлющего каталога *mail*.

Ну а теперь откройте файл *websites.xml* в каталоге */remote* и добавьте в него показанный в примере 7.40 XML-код перед тегом `</WebSites>`.

Пример 7.40. XML-код, добавляемый в файл websites.xml

```
<WebSite name="Outlook Webmail" uri="/mail" imageUrl="/mail/icon.png"
absolute="false"></WebSite>
```

Закончив с конфигурированием, мы должны обеспечить сетевой доступ к сайту на сервере Windows Home Server. Дважды щелкните по значку Windows Home Server Console на рабочем столе. Когда консоль загрузится, нажмите кнопку Настройки (Settings) в правом верхнем углу. В диалоговом окне Настройки выберите в левой панели раздел Удаленный доступ (Remote Access) и нажмите кнопку Включить (Turn On) в правой панели (в предположении, что он был выключен), как показано на рис. 7.14. Если удаленный доступ уже был включен, пропустите этот шаг.

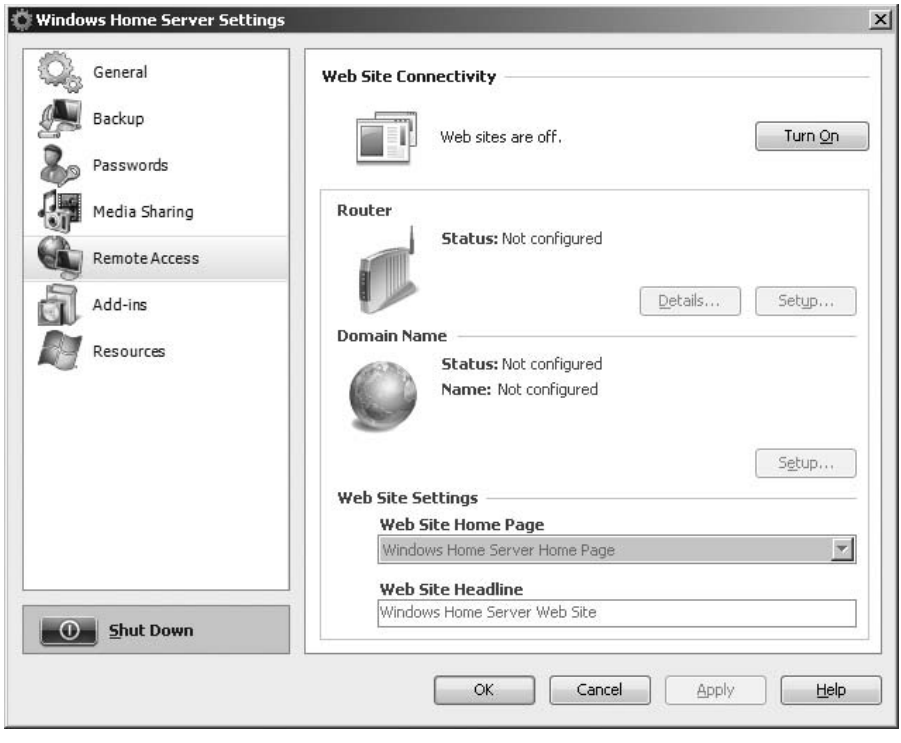

Рис. 7.14. *Настройка удаленного доступа*

Запуск приложения

Убедитесь, что приложение *WHSMailHost* запущено на ПК, где установлен Outlook. Затем зайдите на подразумеваемый по умолчанию сайт на сервере Windows Home Server. После успешного ввода имени и пароля вы должны увидеть справа на экране ссылку Outlook Webmail, как показано на рис. 7.15.

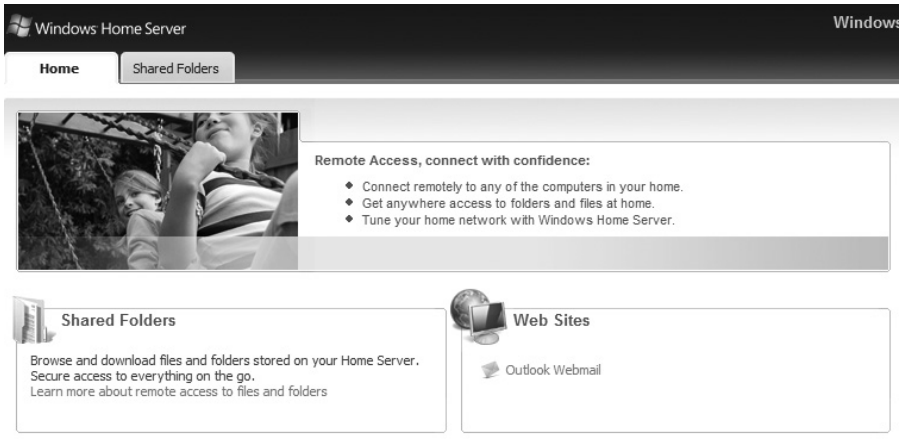

Рис. 7.15. *Значок Outlook Webmail*

Щелкните по этой ссылке, чтобы запустить приложение. Если все сделано правильно, то вы должны увидеть веб-страницу, показанную на рисунке.

Заключительные замечания

Уф! Написав два приложения и одну разделяемую сборку, мы получили веб-надстройку над Windows Home Server, которая обеспечивает удаленный доступ к хранилищу сообщений Outlook. В настоящий момент приложение разрешает только чтение почты и обладает лишь минимальной функциональностью, но уже может служить неплохой отправной точкой для разработки полноценного почтового клиента с веб-интерфейсом, который будет поддерживать вложения и флаги сообщений. Можно даже включить в него календари, контакты и задачи Outlook.

III

Технические устройства

8

Машинка, управляемая с пульта Wiimote

Автор	Брайан Пик
Сложность	Средняя
Необходимое время	5 часов
Стоимость	$40 за пульт Wii Remote (бесплатно, если у вас есть игровая приставка Nintendo), $80 за интерфейсную плату Phidget Interface Kit, $25 за дистанционно управляемую машинку
Программное обеспечение	Visual C# 2008 Express, Visual Basic 2008 Express или любая полная редакция Visual Studio 2008
Оборудование	Пульт Nintendo Wii Remote, интерфейсная плата Phidget Interface Kit, дистанционно управляемая машинка с цифровыми входами, паяльник, припой, провода
Адрес в Интернете	*http://www.c4fbook.com/WiimoteCar*

Продажи игровой приставки Nintendo Wii в Северной Америке начались в ноябре 2006 года, и с тех пор ее революционный пользовательский интерфейс поднял компьютерные игры на новый уровень. За несколькими исключениями предшествующие приставки позволяли управлять игрой только с помощью стандартного пульта и кнопок. Nintendo Wii подарила игрокам пульт Wii Remote, реагирующий на движения. Приверженцы ласково называют его просто *Wiimote* (рис. 8.1).

Рис. 8.1. *Пульт Wii Remote (Wiimote)*

В этом проекте мы воспользуемся сенсорными возможностями Wiimote для дистанционного управления стандартной игрушечной машинкой – водитель сможет крутить руль влево-вправо, просто наклоняя в разные стороны Wiimote.

Краткий обзор

Для тех, кто никогда не видел пульт Wii remote, скажем, что это ручное устройство, похожее на обычный телевизионный пульт ДУ. Оно входит в комплект поставки игровой приставки Nintendo Wii, но продается и отдельно примерно за 40 долларов. Внутри пульта скрыто на удивление много разнообразной аппаратуры:

- пространственный сенсор движения (трехкоординатный акселерометр);
- инфракрасная камера с распознаванием четырех точек;
- семь кнопок;
- джойстик, поддерживающий управление по направлениям вверх-вниз-вправо-влево;
- четыре светодиода;
- динамик;
- вибрационный двигатель;
- порт расширения;
- возможность подключения по Bluetooth.

Последний пункт и делает это небольшое устройство таким привлекательным. Совместимость с Bluetooth позволяет подключить Wiimote

практически к любому компьютеру, оборудованному Bluetooth-интерфейсом.

Помимо пульта Wiimote нам понадобится интерфейсная плата *Phidget Interface Kit*. «Фиджеты» (phidgets) (*http://www.phidgets.com/*) – это подключаемые по USB устройства, позволяющие выполнять простые операции ввода/вывода и обладающие чрезвычайно простым программным интерфейсом. Интерфейсную плату можно использовать для размыкания и замыкания переключателей, для считывания данных с цифровых входов и аналоговых датчиков (например, температуры), а также для выставления показаний датчиков. В этом проекте нас будут интересовать только цифровые выходы интерфейсной платы, с помощью которых мы станем размыкать и замыкать переключатели пульта дистанционного управления машинкой, моделируя действия пользователя, манипулирующего пультом. Продаются разные модели плат Phidget Interface Kit, нас устроит 0/16/16 или 0/0/4.

Для этого проекта нужны три вещи: интерфейс с пользователем (Wiimote), интерфейс с машинкой (оригинальный прилагаемый к ней пульт ДУ) и переходник между тем и другим (Phidget Interface Kit). Программа, которую мы напишем в этой главе, будет считывать данные с Wiimote, определять, как он наклонен, и соответственно включать или выключать сигнал на выходе интерфейсной платы. Последняя будет аппаратно подключена напрямую к оригинальному пульту управления машинкой, поэтому переключение сигнала эквивалентно действиям человека, который держит пульт в руках и манипулирует джойстиками (рис. 8.2).

Определения терминов, относящихся к Wiimote

Акселерометр – электронное устройство, измеряющее проекцию силы (на ось чувствительности устройства), приложенной к его корпусу. Встроенный в Wiimote акселерометр может измерять проекции силы на три координатные оси.

Bluetooth – беспроводной протокол передачи данных на короткие расстояния. В Wiimote этот протокол применяется для подключения к приставке Wii или к персональному компьютеру.

Инфракрасный – диапазон длин волн, невидимых невооруженным взглядом. Сенсорная панель Wii, которая ставится на телевизор, испускает инфракрасные лучи с двух точек. Wiimote «видит» их и таким образом определяет свое положение в пространстве.

Светодиод (LED). В нижней части Wiimote расположены четыре светодиода, с помощью которых можно сообщить простую информацию о состоянии.

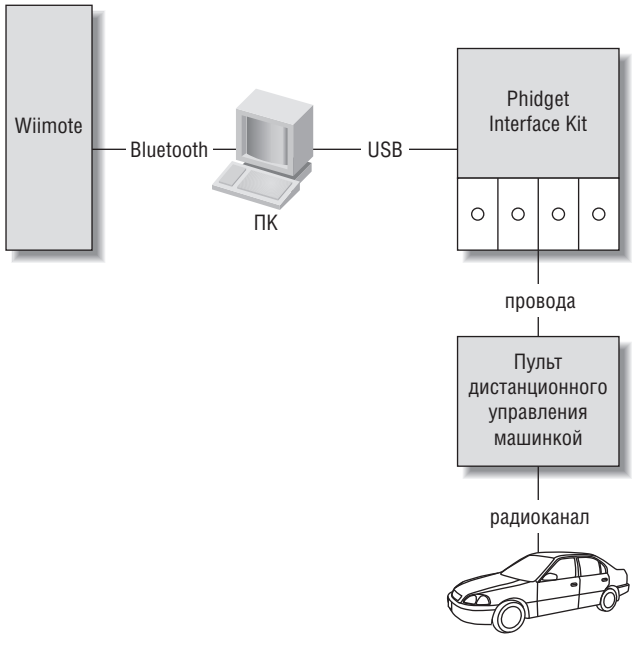

***Рис. 8.2.** Архитектурная диаграмма проекта*

Сборка аппаратуры

Первым делом нам надо как-то подключить дистанционно управляемую машинку к компьютеру. Проще всего припаять к пульту интерфейсную плату Phidget Interface Kit и подключить ее к USB-порту. Самое главное здесь – найти дистанционно управляемую машинку с «цифровым» контроллером. Иными словами, манипулирование джойстиками должно просто размыкать и замыкать некие переключатели. Машинка с «аналоговым» управлением позволяет изменять скорость и поворачивать руль, что, конечно, повышает степень управляемости, но при этом заметно увеличивает сложность проекта и стоимость необходимого для его реализации оборудования. По моему опыту, чем машинка дешевле, тем вероятнее, что она имеет цифровое управление. И еще – если пульт снабжен кнопками, а не джойстиками, то входные сигналы устройства почти наверняка цифровые.

Для проекта из этой главы я взял машинку из коллекции Dub City фирмы Jada Toys примерно за 20 долларов (рис. 8.3). Если точно такой у вас нет, загляните на сайт *http://www.c4fbook.com/* и поинтересуйтесь, с какими еще моделями работали люди, успешно завершившие этот проект.

Рис. 8.3. *Дистанционно управляемая машинка и пульт ДУ*

Если вы планируете устраивать гонки между несколькими машинками, то не забудьте, что у них должны быть разные радиочастоты. Обычно на машинке указана рабочая частота или соответствующая буква, например A/B/C. Если вы купите машинки с разными частотами (или буквами), то все получится.

Для начала открутите все винты на задней крышке пульта и снимите крышку. Внутри находится печатная плата. Если она привинчена к корпусу, отвинтите ее, так чтобы можно было ее вынуть и свободно повернуть. На передней стороне платы, там, где с ней контактируют джойстики, вы увидите несколько простеньких переключателей, которые размыкаются и замыкаются при качании джойстика (рис. 8.4).

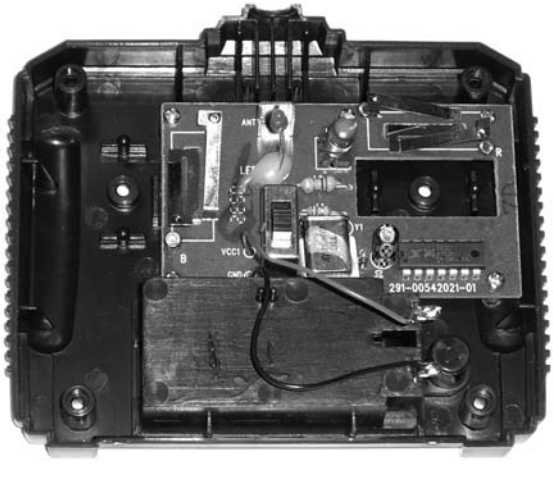

Рис. 8.4. *Пульт ДУ в разобранном состоянии*

Ко всем контактам каждого переключателя нужно припаять провода. Для этого отрежьте восемь кусков провода длиной примерно по 15 см. Подойдет почти любой провод. Если у вас завалялся старый телефонный или сетевой кабель, берите, не раздумывая. Очистите концы проводов от изоляции. Затем припаяйте по одному концу провода к каждому контакту на плате. С каждым направлением связаны два контакта. Если они перемкнуты джойстиком, то цепь замыкается, пульт посылает команду, машинка едет (рис. 8.5).

Припаяв провода к контактам, мы сможем замыкать и размыкать переключатели электронно, имитируя тем самым действия джойстика.

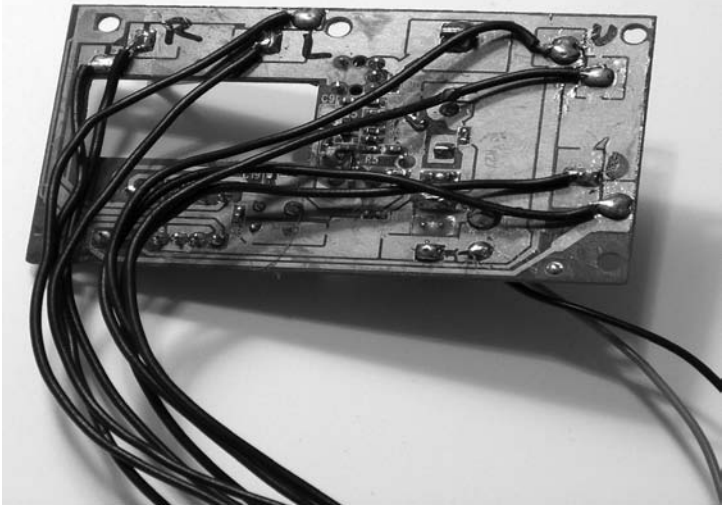

Рис. 8.5. *Пульт с припаянными проводами*

Интерфейсная плата

Припаяв все провода к плате пульта, мы должны вставить свободные концы в клеммы на интерфейсной плате Phidget Interface Kit. Это делается по-разному в зависимости от модели интерфейсной платы.

Будем предполагать, что для всех плат выходные порты должны отображаться на направления движения машинки, как показано в табл. 8.1.

Таблица 8.1. Назначение выходных портов

Выходной порт	Направление движения	Выходной порт	Направление движения
0	Вперед	2	Влево
1	Назад	3	Вправо

Интерфейсная плата Phidget 0/0/4

На этой плате находятся четыре реле, которые можно замыкать и размыкать программно. Перед каждым реле расположена планка с винтовыми клеммами, открывающими доступ к контактам реле. Для каждого реле есть три клеммы, помеченные NO (Normally Open – нормально разомкнутый), NC (Normally Closed – нормально замкнутый) и *X*C (Relay X Common – общая линия заземления), где *X* – число от 0 до 3. «Нормально разомкнутый» означает, что переключатель разомкнут, когда ток не идет. «Нормально замкнутый» означает прямо противоположное – без тока переключатель замкнут, а при подаче тока размыкается. Клемму «Common» можно считать аналогом «земли» для реле.

Мы хотим, чтобы переключатели на пульте ДУ были открыты, пока мы их явно не закроем, поэтому из проводов в каждой паре один следует подключить к клемме NO соответствующего реле, а другой – к клемме Common (рис. 8.6).

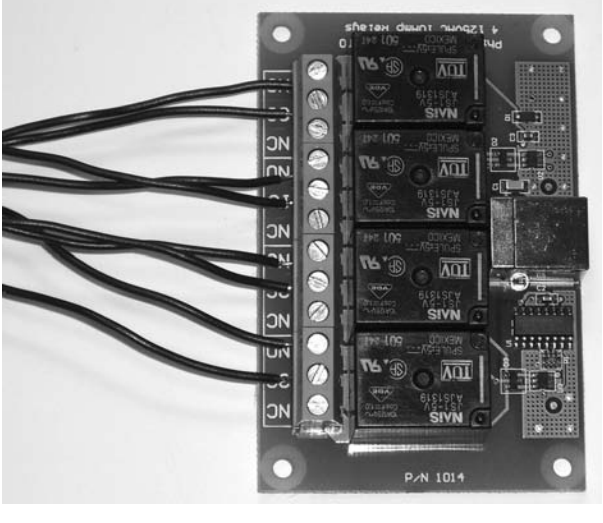

Рис. 8.6. *Интерфейсная плата Phidget 0/0/4 Interface Kit с прикрученными проводами*

Интерфейсная плата Phidget 0/16/16

Эта плата состоит из двух отделов, помеченных Inputs (Входы) и Outputs (Выходы). Отдел Outputs содержит 16 цифровых выходов, пронумерованных от 0 до 15. Есть также восемь клемм заземления, помеченных буквой «G». Соедините один провод из каждой пары с прону-

мерованным выходом, как описано в таблице 8.1, а другой – с клеммой заземления (рис. 8.7).

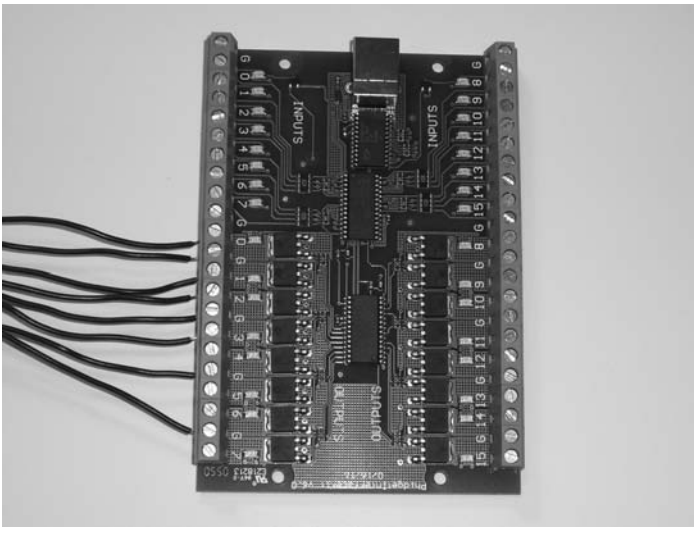

Рис. 8.7. *Интерфейсная плата Phidget 0/16/16 Interface Kit с прикрученными проводами*

Примечание

При использовании платы Phidget Interface Kit 0/16/16 необходим еще один шаг. Подключив провода к обеим платам, вставьте батарейку в пульт ДУ и включите его. Затем вставьте батарейки в машину, переверните ее, так чтобы колеса не касались земли, и тоже включите. Поскольку у обеих плат общая земля, то может оказаться, что некоторые функции машинки «залипли» в положении «включено». Например, машинка едет вперед или колеса заблокированы в одном положении. В таком случае просто поменяйте местами провода на соответствующих этой функции клеммах Phidget Interface Kit, и она «отлипнет».

Разработка программного обеспечения

Написать программную поддержку для этого проекта сравнительно несложно благодаря исключительной простоте API, предоставляемого библиотеками Wiimote и Phidget.

Требования к ПО

Необходимо будет скачать и установить несколько библиотек. Во-первых, чтобы приложение могло взаимодействовать с интерфейсной платой Phidget Interface Kit, понадобится библиотека Phidget API. Она

часто обновляется, и иногда новые версии оказываются несовместимы с предыдущими. Поэтому на сайте *http://www.c4fbook.com/WiimoteCar* выложена версия, которой мы пользовались в этом проекте.

Еще необходима библиотека Managed Wiimote library. Код в этой главе был написан с использованием версии 1.5.2, которую можно скачать со страницы *http://www.codeplex.com/WiimoteLib*.

Реализация ПО

Создайте новый проект типа Windows Application на предпочитаемом вами языке – C# или Visual Basic. В проект нужно будет добавить несколько ссылок. Во-первых, коль скоро приложение будет работать с пультом Wiimote, понадобится ссылка на сборку *WiimoteLib.dll*. Скопируйте файл *WiimoteLib.dll* из дистрибутива в локальный каталог проекта, чтобы при добавлении ссылки его было проще найти. Кроме того, добавьте ссылку на сборку *Phidget21.NET.dll* (рис. 8.8).

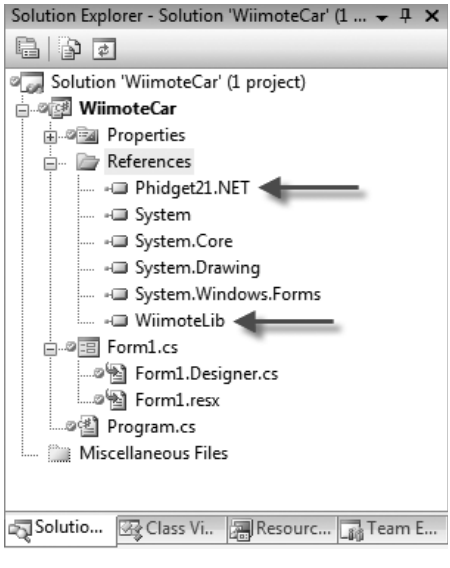

***Рис. 8.8.** Добавление ссылок в проект*

Примечание

Если вы работаете с 64-разрядной версией Windows, то сборок Phidget может не оказаться в глобальном списке доступных сборок. В таком случае перейдите в каталог *C:\Program Files (x86)\Phidgets* и выберите файл *Phidget21.NET.dll* вручную.

Затем откройте файл *Form1.cs* в режиме конструирования и дважды щелкните в любой точке, чтобы создать пустой обработчик события `Load`. Перед тем как вставлять в него код, необходимо добавить в форму

пространства имен и завести переменные-члены для хранения объектов, представляющих Phidget Interface Kit и Wiimote. Код импорта пространств имен показан в примерах 8.1 и 8.2.

Пример 8.1. Импорт пространств имен (версия на C#)

```
using Phidgets;
using WiimoteLib;
```

Пример 8.2. Импорт пространств имен (версия на VB)

```
Imports Phidgets
Imports WiimoteLib
```

Затем добавьте приведенные в примерах 8.3 и 8.4 фрагменты для определения нужных нам переменных-членов. Этот код должен находиться на уровне самой формы.

Пример 8.3. Определение переменных-членов (версия на C#)

```
// объект, представляющий пульт Wiimote
private Wiimote _wiimote;

// объект, представляющий плату Phidget Interface Kit
private InterfaceKit _interfaceKit;
```

Пример 8.4. Определение переменных-членов (версия на VB)

```
' объект, представляющий пульт Wiimote
Private _wiimote As Wiimote

' объект, представляющий плату Phidget Interface Kit
Private _interfaceKit As InterfaceKit
```

Определив переменные-члены, можно вернуться к сгенерированному обработчику события `Load` и включить в него код, который создаст объект `InterfaceKit`, откроет его и будет ждать подключения интерфейсной платы (примеры 8.5 и 8.6).

Пример 8.5. Создание и открытие объекта InterfaceKit (версия на C#)

```
// создать объект InterfaceKit и открыть доступ к плате
_interfaceKit = new InterfaceKit();
_interfaceKit.open();
_interfaceKit.waitForAttachment(5000);
```

Пример 8.6. Создание и открытие объекта InterfaceKit (версия на VB)

```
' создать объект InterfaceKit и открыть доступ к плате
_interfaceKit = New InterfaceKit()
_interfaceKit.open()
_interfaceKit.waitForAttachment(5000)
```

Последнее, что осталось сделать в методе загрузки формы, – создать и сконфигурировать объект `Wiimote`. Библиотека Wiimote предлагает очень простую событийно-ориентированную модель соединения с пуль-

том Wiimote. Достаточно создать объект, установить обработчик события поступления данных от пульта и соединиться с пультом. Вся процедура приведена в представленном ниже фрагменте кода, который следует включить все в тот же метод `Load`.

Пример 8.7. Создание объекта Wiimote (версия на C#)

```
// создать объект Wiimote
_wiimote = new Wiimote();

// установить обработчик события, который будет вызван,
// когда Wiimote отправит пакет данных
_wiimote.WiimoteChanged += Wiimote_WiimoteChanged;

// соединиться с Wiimote
_wiimote.Connect();

// принимать данные только от кнопок и акселерометра
_wiimote.SetReportType(InputReport.ButtonsAccel, true);
```

Пример 8.8. Создание объекта Wiimote (версия на VB)

```
' создать объект Wiimote
_wiimote = New Wiimote()

' установить обработчик события, который будет вызван,
' когда Wiimote отправит пакет данных
AddHandler _wiimote.WiimoteChanged, AddressOf Wiimote_WiimoteChanged

' соединиться с Wiimote
_wiimote.Connect()

' принимать данные только от кнопок и акселерометра
_wiimote.SetReportType(InputReport.ButtonsAccel, True)
```

Теперь осталось только написать сам обработчик события, который будет принимать данные от Wiimote и менять состояние соответствующих выходов на плате Phidget Interface Kit, управляя тем самым машинкой.

Создайте обработчик события, как показано в примерах 8.9 и 8.10.

Пример 8.9. Реализация обработчика события Wiimote_WiimoteChanged (версия на C#)

```
void Wiimote_WiimoteChanged(object sender, WiimoteChangedEventArgs e)
{
  // получить текущее состояние Wiimote
  WiimoteState ws = e.WiimoteState;

  // если нажата кнопка 1, изменить выход Вперед
  _interfaceKit.outputs[0] = ws.ButtonState.One;

  // если нажата кнопка 2, изменить выход Назад
  _interfaceKit.outputs[1] = ws.ButtonState.Two;

  // если Wiimote достаточно сильно наклонен влево, изменить выход Влево
```

```
    _interfaceKit.outputs[2] = (ws.AccelState.Values.Y < -0.07f);

    // если Wiimote достаточно сильно наклонен вправо, изменить выход Вправо
    _interfaceKit.outputs[3] = (ws.AccelState.Values.Y > 0.07f);
}
```

Пример 8.10. Реализация обработчика события Wiimote_WiimoteChanged (версия на VB)

```
Private Sub Wiimote_WiimoteChanged(ByVal sender As Object, ByVal e As
WiimoteChangedEventArgs)
  ' получить текущее состояние Wiimote
  Dim ws As WiimoteState = e.WiimoteState

  ' если нажата кнопка 1, изменить выход Вперед
  _interfaceKit.outputs(0) = ws.ButtonState.One

  ' если нажата кнопка 2, изменить выход Назад
  _interfaceKit.outputs(1) = ws.ButtonState.Two

  ' если Wiimote достаточно сильно наклонен влево, изменить выход Влево
  _interfaceKit.outputs(2) = (ws.AccelState.Values.Y < -0.07f)

  ' если Wiimote достаточно сильно наклонен вправо, изменить выход Вправо
  _interfaceKit.outputs(3) = (ws.AccelState.Values.Y > 0.07f)
End Sub
```

Этот обработчик анализирует объект `WiimoteState`, переданный в составе аргументов события, и устанавливает состояния выходов платы Phidget Interface Kit. Если нажата кнопка 1, то подается сигнал на выход 0, а если нажата кнопка 2, то на выход 1. Если пульт Wiimote наклонен влево или вправо на угол, больший заданной пороговой величины, подается сигнал на второй или третий выход соответственно.

Если реле замкнуто, то переключатель на пульте ДУ замкнут. Когда этот переключатель замкнут, машинке посылается сигнал, и она совершает те или иные перемещения.

Возможно, заданные нами пороги вам покажутся слишком чувствительными (или недостаточно чувствительными). Изменив значения, с которыми сравнивается свойство `AccelState.Values.Y` (в нашем примере 0.07f), вы сможете увеличить или уменьшить порог и создать комфортные для себя условия.

Дополнительные действия для пользователей 64-разрядных версий Windows

Те, кто работает с 64-разрядной версией Windows, должны предпринять еще один шаг. Библиотеки Phidget не полностью совместимы с 64-разрядными редакциями Windows, но эту трудность можно обойти, создав конфигурацию для платформы x86 (32-разрядной) и собрав 32-разрядное приложение.

Для этого выберите из меню пункт Build→Configuration Manager... (рис. 8.9).

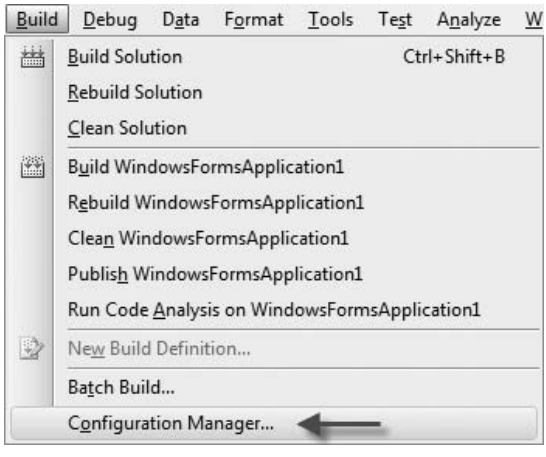

Рис. 8.9. *Пункт меню Configuration Manager...*

В диалоговом окне Configuration Manager выберите строку <New...> из выпадающего списка Active solution platform (Активная платформа решения) в правом верхнем углу (рис. 8.10).

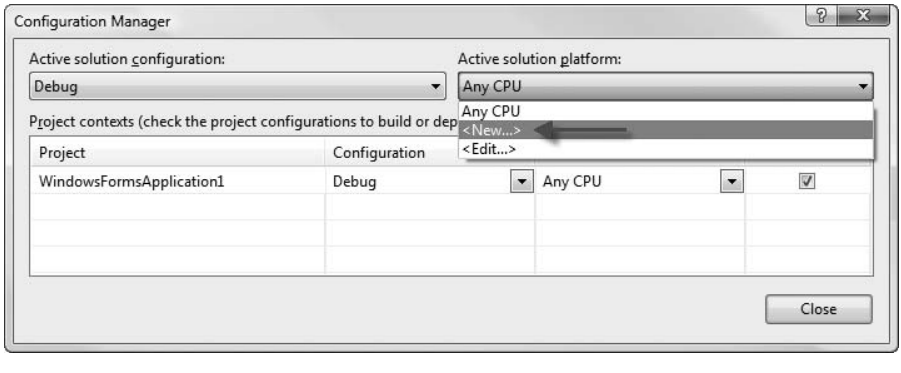

Рис. 8.10. *Диалоговое окно Configuration Manager...*

В окне New Solution Platform выберите из верхнего выпадающего списка пункт x86, а из нижнего – Any CPU (рис. 8.11).

Закройте все окна и убедитесь, что на стандартной панели инструментов в качестве целевой платформы выбрана x86 (рис. 8.12).

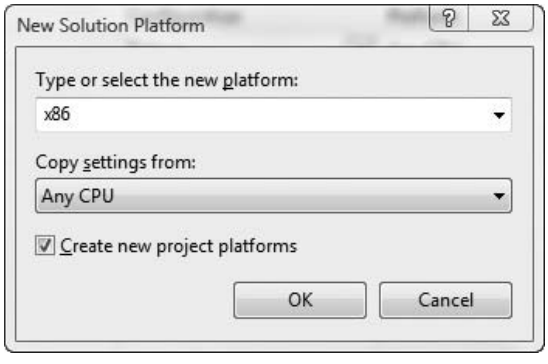

Рис. 8.11. *Диалоговое окно New Solution Platform*

Рис. 8.12. *x86 в качестве целевой платформы на стандартной панели инструментов*

Использование приложения

Пора бы уже и повеселиться, управляя машинкой с помощью пульта Wiimote. Но сначала все-таки еще несколько шагов.

Сопряжение Wiimote

К сожалению, этот шаг в значительной степени зависит от адаптера и стека Bluetooth, установленного на вашем компьютере, поэтому точные инструкции дать невозможно. Кроме того, хотя пульт Wiimote и поддерживает протокол Bluetooth, он не является 100-процентно совместимым устройством, поэтому не исключено, что с тем адаптером и стеком, которые стоят на вашем компьютере, он вообще откажется работать. По крайней мере, убедитесь, что установлены самые последние версии драйверов Bluetooth-устройства.

Для сопряжения Wiimote выполните следующие действия:

1. Запустите программное обеспечение Bluetooth и заставьте его искать устройство.
2. Нажмите и удерживайте кнопки 1 и 2 на пульте. Нижние светодиоды должны замигать. Не отпускайте кнопки, пока процедура не дойдет до конца.
3. В списке найденных устройств должно появиться название Nintendo RVL-CNT-01. Если не появилось, попробуйте еще раз с самого начала.

4. Нажмите кнопку Next (Далее) для перехода к следующему шагу мастера. Если на каком-то этапе вас попросят ввести код защиты или PIN-код, оставьте поле пустым или нажмите кнопку Skip (Пропустить). Никакое число не вводите.
5. Возможно, будет задан вопрос, какую службу Wiimote использовать. В таком случае выберите службу keyboard/mouse/HID (никакой другой и быть не должно).
6. Нажмите кнопку Finish (Готово).

Светодиоды должны по-прежнему моргать, а в списке подключенных Bluetooth-устройств должен появиться ваш пульт.

Установка платы Phidget Interface Kit

Просто воткните плату Phidget Interface Kit в любой USB-порт, она должна опознаться как USB-устройство типа Human Interface Device (HID) (проще говоря, устройство ввода), не требующее специальных драйверов.

Управление машинкой

Ну а теперь, наконец, вставьте батарейки в пульт и в машинку и включите их. Запустите написанное только что приложение. При старте оно подключится к пульту Wiimote и к плате Phidget. Окно программы, надо признать, скучновато, поскольку в него ничего не выводится, однако смысл работы приложения – управление Wiimote и машинкой, а не красивые Windows-формы.

Держите пульт в обеих руках кнопками вверх параллельно земле, так чтобы джойстик был слева, а кнопки 1 и 2 справа (см. рис. 8.13).

Чтобы машинка поехала вперед, нажмите кнопку 1, чтобы назад – кнопку 2. Для поворота наклоняйте (но не вращайте!) пульт влево или вправо.

Рис. 8.13. *Так надо держать пульт Wiimote для управления машинкой*

Заключительные замечания

Вот мы и добились своего – получили машинку, дистанционно управляемую с пульта Wiimote. Для этого потребовались пульт, машинка, интерфейсная плата Phidget Interface Kit и немного времени, проведенного с паяльником.

Но не останавливайтесь на этом. Приложив немного усилий, вы найдете множество способов сделать этот проект еще интереснее. Например, можно изменить схему управления – вместо нажатия на кнопки 1 и 2 отслеживать перемещение пульта вперед и назад. Или, добавив беспроводную веб-камеру, сделать шпионскую машину, ведущую скрытую съемку.

В следующей главе мы покажем, как можно использовать пульт Wiimote совсем для другой цели.

9

Электронная доска и Wiimote

Автор	Джонни Ли (*http://procrastineering.blogspot.com/*), Бун Джин Го (*http://www.boonjin.com/*), Майк Корчински и Брайан Пик
Сложность	Средняя
Необходимое время	3 часа
Стоимость	$40 за пульт Wii Remote (бесплатно, если у вас есть игровая приставка Nintendo), плюс пара долларов на инфракрасный светодиод, резисторы и маркер-ластик
Программное обеспечение	Visual C# 2008 Express, Visual Basic 2008 Express или любая полная редакция Visual Studio 2008
Оборудование	Пульт Nintendo Wii Remote, инфракрасный (ИК) светодиод, нормально разомкнутый быстродействующий переключатель, батарейка 1.5 В, маркер-ластик, паяльник, припой, провода
Адрес в Интернете	*http://www.c4fbook.com/WiimoteWhiteboard*

Краткий обзор

В предыдущей главе вы уже имели случай убедиться в том, что пульт Wii Remote (Wiimote) – весьма гибкое устройство, обладающее рядом интересных сенсорных возможностей и к тому же без труда подключаемое к ПК. В этой главе мы покажем, как создать интерактивную электронную доску с несколькими указками, используя встроенную в переднюю панель пульта камеру.

Здесь мы будем рассматривать только детали реализации, относящиеся непосредственно к электронной доске. Предполагается, что вы уже подключили пульт Wiimote к компьютеру, оборудованному Bluetooth-адаптером. Если вы не знаете, как это сделать, ознакомьтесь с инструкциями в главе 8.

Определение интерактивной электронной доски

Появившаяся в школах в 1801 году, обычная меловая доска заслуженно признается одним из самых мощных и важных образовательных инструментов. Она позволяет лектору легко и быстро представлять визуальный материал большой аудитории, просто изображая его на поверхности, позволяющей стирать и записывать снова. Два века спустя значок меловой доски стал общепризнанным символом профессии учителя. По мере развития технологических и производственных возможностей меловая доска принимала различные формы. Но ее принципиальное назначение не изменилось. Современные доски лишь стали гипоаллергенными, беспыльными, легко стирающимися и обеспечивающими высокую четкость. Доска – это обязательный атрибут любого современного офиса и многих учебных кабинетов. Однако, как и бумага, она представляет собой пассивную поверхность. Написанный материал статичен. Его нельзя переместить, скопировать, сохранить и восстановить позднее. Написанная информация остается на поверхности. Стертая – пропадает навсегда.

Интерактивная доска, которую часто называют также цифровой, – это попытка объединить достоинства работы с цифровой информацией, например возможность копировать, вставлять, сохранять, открывать и посылать по почте, с физическими возможностями настоящей доски по презентации, совместному использованию и рисованию визуально воспринимаемой информации. На такой доске не пишут маркером. Вместо этого применяемая сенсорная технология распознает положения стилоса, а компьютер с помощью виртуальных чернил имитирует рисование. Поскольку информация представлена в цифровом виде, ее можно сохранять, распечатывать, отправлять по почте и стирать одним нажатием на кнопку. Стилос может рисовать любым цветом, любым размером шрифта, имитировать кисть, карандаш или пульверизатор. Он позволяет выделять и перетаскивать объекты, как курсор мыши. Многие коммерческие интерактивные доски даже имитируют курсор мыши, давая возможность взаимодействовать с любыми программами, например графическими редакторами, веб-броузерами, симуляторами или играми.

Это позволяет преподавателю управлять компьютером, стоя перед классом и касаясь поверхности доски указкой; включать в презентации картинки, графики и видео; писать замечания и рисовать на экране стилосом; сохранять учебные материалы в цифровом виде; демонстрировать интерактивные программы. В компаниях интерактивная электронная

доска используется в качестве инструмента для организации совместной работы сотрудников, не находящихся в одном офисе, а также для документирования технологических процессов. Цифровые метки можно передавать через Интернет, что создает у удаленных участников ощущение общего владения информацией. Проектировщики могут подгружать необходимые цифровые материалы, манипулировать документами и создавать аннотации, которые легко архивируются.

К сожалению, интерактивная доска может стоить очень дорого, цена зачастую доходит до нескольких тысяч долларов, не считая проектора или компьютера. В результате лишь учреждения с очень большим бюджетом на оборудование могут позволить себе такую роскошь. Понятно, что для большинства физических лиц вопрос о приобретении такой системы даже не возникает. Однако в этой главе вы узнаете, как создать эффективную интерактивную доску, имея лишь пульт Nintendo Wii Remote и кое-какие электронные компоненты на сумму 10 долларов. И все это будет работать практически на любом компьютерном или телевизионном экране, так что проектор необязателен.

Использование Wii Remote с инфракрасной камерой

Для интерактивного создания изображения необходимо отслеживать положение стилоса. Один из способов добиться этого – воспользоваться камерой. Камера видит, в какой точке изображения находится стилос, а мы можем получить эти координаты в пикселах и сообщить компьютеру, где должен находиться курсор на экране. На рынке существует несколько интерактивных сенсорных систем ввода на базе камеры, в том числе Microsoft Surface, дисплей Perceptive Pixel и продукты компании Smart Technologies. Нередко в них используется почти инфракрасная (ИК) подсветка точек касания. «Почти инфракрасная» означает, что длина световой волны находится на красном конце спектра за пределами человеческого восприятия. ИК-излучение часто применяется в пультах дистанционного управления телевизорами и стереопроигрывателями. Поскольку этот свет для нас невидим, следящая система с камерой может применять специальную ИК-подсветку для выделения тех или иных объектов.

За черным пластиком на торце каждого пульта Wiimote находится инфракрасная камера. Именно она, следя за сенсорной панелью, которая находится рядом с вашим телевизором, наделяет приставку Wii свойствами указки. Название «сенсорная панель» только вводит в заблуждение, поскольку никаких сенсоров там нет, а есть две группы ИК-излучателей. На изображении, получаемом от камеры, они выглядят как две яркие точки. Отыскав эти точки, приставка понимает, на какую точку экрана вы указываете.

Микросхему с камерой производит компания PixArt Imaging, Inc. У нее есть интересная особенность – она порождает не изображение, а лишь информацию о координатах ярких точек, общим числом не более четырех. Эту микросхему компания называет интегрированной «Системой слежения за несколькими объектами» (Multi-Object Tracking System – MOTS). Это аппаратный процессор машинного зрения, который отслеживает перемещение ярких точек. Если бы вы стали писать собственную систему машинного зрения, ориентированную на обычную веб-камеру, то должны были бы перебрать все пиксели в каждом кадре, пометить те, для которых яркость особенно велика, сгруппировать помеченные пиксели в кластеры и усреднить координаты последних, чтобы найти центр. Этот алгоритм называется «отслеживанием пятен» (blob tracking). К счастью, всю эту работу уже проделала за нас микросхема. Но это также означает, что исходное изображение с камеры недоступно управляющему компьютеру по каналу Bluetooth.

Возвращаемые камерой данные слежения – это изображение с разрешением 1024×768 пикселов и частотой обновления 100 кадров в секунду. Камера способна одновременно отслеживать местоположение не более четырех ИК-точек. Если она видит больше четырех точек, то следит только за четырьмя самыми яркими. Ее можно было бы использовать для отслеживания более четырех точек при мерцающем освещении, но для этого потребовалось бы более сложное программное обеспечение.

ИК-перо

Поскольку камера на пульте Wiimote чувствительна к инфракрасным источникам, то для отслеживания положения стилоса на экране нам понадобится стилос, испускающий инфракрасный свет.

Для сборки такого стилоса потребуется несколько компонентов:

- инфракрасный светодиод;
- нормально разомкнутый быстродействующий переключатель;
- батарейка типа AA;
- тонкая проволока (калибра 26 или выше);[1]
- корпус, где все это размещается; подойдет маркер-ластик.

Сборка пера – это одновременно и наука, и искусство. Если вы полагаете, что будете возвращаться к этому проекту часто – и для практического использования, и для совершенствования, то потратьте время, чтобы аккуратно скомпоновать всю электронику в корпусе маркера, как показано на рис. 9.1.

[1] Американский калибр 26 соответствует диаметру провода 0,4049 мм. Чем калибр больше, тем тоньше провод. – *Примеч. перев.*

Рис. 9.1. *ИК-перо, собранное в корпусе, сделанном из маркера-ластика*

Но если в тесной полости маркера все не помещается, поищите более просторные корпуса. Например, чуть больше места найдется в трубочке для компьютерных деталей. А вместо крышечки в нее можно вставить кончик от маркера, чтобы было ощущение стилоса (см. рис. 9.2).

Такие трубочки можно приобрести на сайте Amazon и в других розничных магазинах. Они часто продаются вместе с ремкоплектом для ПК, так что, возможно, в вашем хозяйстве такая уже есть. Поскольку разместить схему в подобной трубочке совсем просто, будем иметь ее в виду как запасной вариант, а основное внимание уделим сборке пера в корпусе маркера.

Рис. 9.2. *ИК-перо, собранное в трубочке для компьютерных деталей*

Сборка схемы

Саму схему собрать несложно. Имея паяльник, вы быстро с этим справитесь. Если вы раньше никогда не паяли, почитайте какие-нибудь руководства в Интернете.

Но сначала подумайте, как будет расположена кнопка относительно ИК-светодиода. В идеале она должна быть как можно ближе к кончику пера, потому что это наиболее естественно для пользующегося указкой. Стоит сделать провода немного длиннее, чем необходимо, чтобы можно было подправить схему перед вставкой батарейки. Ведь видеть, что делаешь, гораздо удобнее, чем действовать на ощупь.

Рассмотрим, из чего будет состоять наша схема. Светодиоды бывают самых разных цветов и продаются в любом современном магазине бытовой электроники. Вполне возможно, что у вас под рукой уже есть несколько штук. Инфракрасные светодиоды тоже найти нетрудно – в Интернет-магазинах или в местной лавке, торгующей электронными компонентами. Быть может, вам удастся вытащить такой из старого пульта ДУ. Хотя работать будут многие светодиоды, мы рекомендуем выбрать Vishay TSAL 6400 – мощный ИК-светодиод с номинальным током 100 мА. Его продают многие Интернет-магазины, например Mouser Electronics (*http://www.mouser.com/*). Поищите на сайте деталь с номером TSAL 6400, стоит она примерно 30 центов. Поскольку мы собираемся использовать батарейку AAA, дающую напряжение 1.5 В, то нужен светодиод с прямым напряжением не более 1.7 В. В этом случае можно будет обойтись без резистора, стало быть, придется втискивать в корпус на одну деталь меньше и ровно на одну деталь меньше паять. Кроме того, понадобится нормально разомкнутый быстродействующий кнопочный переключатель. Подбирая его, не забывайте о размере корпуса. Подойдет что-нибудь типа Mountain Switch 10PA 019, его тоже можно найти на сайте Mouser. Ну и, наконец, прикрутите провода к батарейке изолентой или подыщите какой-нибудь батарейный отсек. В корпус пера войдет, например, такой отсек для батареек AAA – Keystone Electronics 137, Mouser Part 534-137.

Познакомившись с необходимыми компонентами, мы можем спаять простенькую схему, которая будет включать светодиод при нажатии кнопки. Это позволит нам щелкать и перетаскивать объекты по экрану. Электрическая схема показана на рис. 9.3.

Когда переключатель включен, цепь замкнута, по ней идет ток, и светодиод загорается. Если отпустить кнопку, то цепь размыкается, и светодиод гаснет. Важно помнить, что у светодиода два полюса, поэтому работать он будет только, если провода подсоединены с соблюдением полярности. Анод должен быть соединен с плюсом батарейки, а катод – с минусом. В документации по светодиоду должно быть указано, где анод, а где катод. Впрочем, отметим, что более длинный провод или закругленная (не плоская) сторона почти всегда указывают на анод.

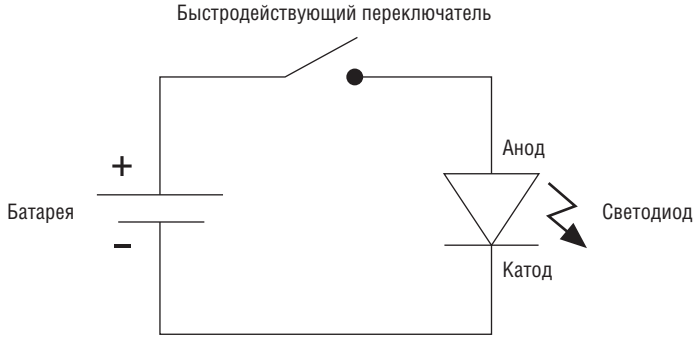

Рис. 9.3. *Схема ИК-пера*

Протестируйте схему, прежде чем вставлять ее в корпус, чтобы сэкономить время и не рвать потом на себе волосы.

Примечание

Чтобы протестировать правильность подключения светодиода и работоспособность пера, воспользуйтесь тем, что многие сотовые телефоны с камерой способны различать инфракрасное излучение. Направьте камеру на светодиод и нажатием кнопки активизируйте перо. Если все работает, то перо должно светиться на экране телефона.

Подготовка маркера

Прежде чем приступать к окончательной сборке, подготовим маркер. С помощью остроносых плоскогубцев снимите заднюю крышечку и вытащите внутреннюю начинку. Крышечку не выбрасывайте, она еще пригодится. Теперь у вас есть пустой корпус маркера, чистый и сухой. Просверлите или прорежьте сбоку дырочку для переключателя, но не слишком большую, чтобы переключатель не проваливался. Для этой цели можно воспользоваться дремелем, но подойдет и обычная дрель и даже ножичек со сменными лезвиями (осторожно, не порежьтесь!). Попутно можете рассверлить кончик пера, погрузив дремель внутрь. Это сильно упростит задачу просовывания светодиода сквозь кончик. Кстати, не забудьте снять с переключателя шайбу и гайку – сейчас самое время.

Теперь все готово к сборке ИК-пера. Это будет испытанием на терпение и усидчивость, в чем-то схожим с игрой Operation. Вам нужно придать схеме такую форму, чтобы протолкнуть ее вглубь корпуса. Не торопитесь, а то оторвете провод от переключателя или от светодиода. Возможно, потребуется несколько попыток, прежде чем удастся точно совместить светодиод с кончиком пера. Только не применяйте силу. Когда светодиод дойдет до узкой части корпуса, начинайте потихоньку

проталкивать кнопку через заранее проделанное отверстие. Для этого можно изготовить из проволоки петельку, подцепить ею кнопку и попытаться вытянуть ее наружу. Пинцет и плоскогубцы для этой цели слишком грубые орудия. Когда переключатель займет свое место, зафиксируйте его шайбой и гайкой. Применяя описанную выше методику, протестируйте схему. Проще найти отпаявшийся контакт сейчас, чем после того, как вы уложите все провода внутри корпуса и закроете его крышечкой. Если схема работает нормально, поместите внутрь пера остаток проводов и батарею (с отсеком или без).

Теперь, когда у нас есть работающее ИК-перо, настало время написать программу, которая будет управлять курсором мыши.

Подготовка проекта

Создайте новый проект типа Windows Application и назовите его «WiimoteWhiteboard». При этом будет создан пустой файл *Form1.cs* или *Form1.vb*. Программа будет состоять из трех частей: главная форма, содержащая графический интерфейс пользователя и координирующая работу прочих частей программы, форма для калибровки пера и класс для выполнения кое-каких математических вычислений. Мы начнем с описания двух более простых компонентов, а потом перейдем к главной форме. Важно представлять себе, каковы отдельные части, прежде чем соединять их вместе.

Преобразование координат камеры

В предыдущей главе вы научились писать программу, которая читает данные от акселерометров пульта Wiimote. Чтобы получить значения координат ИК-сенсора Wiimote, можно воспользоваться массивом `WiimoteState.IRState.IRSensors`, передаваемым обработчику события `WiimoteChangedEventHandler`. Координаты каждой видимой точки вы получите, но они определяют положение точек в системе координат камеры. Если Wiimote не расположен точно напротив дисплея, так что все пиксели точно совпадают, то необходимо преобразовать полученные от камеры координаты в систему координат дисплея. Точнее, необходимо установить соответствие между пикселами камеры и пикселами дисплея. Для этого мы используем плоское проективное преобразование (планарную гомографию).

Когда вы смотрите на мир через объектив камеры, то воспринимаете его как плоское изображение, искаженное в соответствии с законами перспективы. Удаленные объекты кажутся маленькими, а квадраты выглядят как трапеции. Если вы смотрите на плоскую поверхность, то это искажение можно описать матрицей проективного преобразования. Именно так мы и поступим для приведения системы координат камеры к системе координат дисплея.

Строгое объяснение того, как и почему именно так устроено это преобразование, выходит за рамки данной главы. Теоретическое обоснование предоставляет один из разделов математики – линейная алгебря. Для написания программы вам требуется знать всего восемь пар чисел: координаты углов известного прямоугольника на дисплее, или «конечной четверки»: *[(x1,y1), (x2,y2), (x3,y3), (x4,y4)]* – и координаты соответствующих точек на изображении, полученном от камеры, или «исходной четверки»: *[(x1',y1'), (x2',y2'), (x3',y3'), (x4',y4')]*. Этих данных достаточно для построения матрицы преобразования. Таким образом, после включения пера камера будет видеть точку и сможет вычислить, где она должна располагаться на экране.

Приведенный ниже код – это стандартный алгоритм вычисления матрицы плоского проективного преобразования 4×4. Вообще-то для планарной гомографии достаточно и матрицы 3×3, однако матрица 4×4 может быть использована в конвейерах рендеринга, например в системах OpenGL или DirectX, для аппаратного ускорения преобразований более сложных сцен. В нашем проекте это необязательно, но повышает полезность кода. Дополнительную информацию об алгоритмических основах проективных преобразований можете почерпнуть в каком-нибудь учебнике по машинному зрению.

Для начала создайте в проекте класс `Warper`. В нем заведите переменные для исходной четверки, конечной четверки, двух рабочих матриц и окончательной матрицы преобразования, как показано в примерах 9.1 и 9.2. Флаг `dirty` говорит о том, что данные изменились, и матрицу следует вычислить заново.

Пример 9.1. Переменные-члены класса Warper (версия на C#)

```
public class Warper
{
  private float[] _srcX = new float[4];
  private float[] _srcY = new float[4];
  private float[] _dstX = new float[4];
  private float[] _dstY = new float[4];
  private float[] _srcMat = new float[16];
  private float[] _dstMat = new float[16];
  private float[] _warpMat = new float[16];
  private bool _dirty;
}
```

Пример 9.2. Переменные-члены класса Warper (версия на VB)

```
Public Class Warper
  Private _srcX(3) As Single
  Private _srcY(3) As Single
  Private _dstX(3) As Single
  Private _dstY(3) As Single
  Private _srcMat(15) As Single
  Private _dstMat(15) As Single
```

```
  Private _warpMat(15) As Single
  Private _dirty As Boolean
End Class
```

Для инициализации объекта `Warper` тождественной матрицей, то есть такой, где числа на главной диагонали (из левого верхнего угла в правый нижний) равны 1, а все остальные 0, сделаем исходные и конечные точки одинаковыми (расположенными в углах единичного квадрата). В результате рабочие и окончательная матрица будут тождественными (см. примеры 9.3 и 9.4).

Пример 9.3. Инициализация класса Warper (версия на C#)

```
public Warper()
{
  SetIdentity();
}

public void SetIdentity()
{
  SetSource(0.0f, 0.0f,
            1.0f, 0.0f,
            1.0f, 1.0f,
            0.0f, 1.0f);
  SetDestination(0.0f, 0.0f,
                 1.0f, 0.0f,
                 1.0f, 1.0f,
                 0.0f, 1.0f);
  ComputeWarp();
}

public void SetSource( float x0, float y0, float x1, float y1,
                       float x2, float y2, float x3, float y3)
{
  _srcX[0] = x0;
  _srcY[0] = y0;
  _srcX[1] = x1;
  _srcY[1] = y1;
  _srcX[2] = x2;
  _srcY[2] = y2;
  _srcX[3] = x3;
  _srcY[3] = y3;
  _dirty = true;
}

public void SetDestination( float x0, float y0, float x1, float y1,
                            float x2, float y2, float x3, float y3)
{
  _dstX[0] = x0;
  _dstY[0] = y0;
  _dstX[1] = x1;
  _dstY[1] = y1;
  _dstX[2] = x2;
```

```
    _dstY[2] = y2;
    _dstX[3] = x3;
    _dstY[3] = y3;
    _dirty = true;
  }
```

Пример 9.4. Инициализация класса Warper (версия на VB)

```
Public Sub New()
  SetIdentity()
End Sub

Public Sub SetIdentity()
  SetSource(0.0f, 0.0f, 1.0f, 0.0f, 1.0f, 1.0f, 0.0f, 1.0f)
  SetDestination(0.0f, 0.0f, 1.0f, 0.0f, 1.0f, 1.0f, 0.0f, 1.0f)
  ComputeWarp()
End Sub

Public Sub SetSource( ByVal x0 As Single, ByVal y0 As Single, _
                      ByVal x1 As Single, ByVal y1 As Single, _
                      ByVal x2 As Single, ByVal y2 As Single, _
                      ByVal x3 As Single, ByVal y3 As Single)
  _srcX(0) = x0
  _srcY(0) = y0
  _srcX(1) = x1
  _srcY(1) = y1
  _srcX(2) = x2
  _srcY(2) = y2
  _srcX(3) = x3
  _srcY(3) = y3
  _dirty = True
End Sub

Public Sub SetDestination( ByVal x0 As Single, ByVal y0 As Single, _
                           ByVal x1 As Single, ByVal y1 As Single, _
                           ByVal x2 As Single, ByVal y2 As Single, _
                           ByVal x3 As Single, ByVal y3 As Single)
  _dstX(0) = x0
  _dstY(0) = y0
  _dstX(1) = x1
  _dstY(1) = y1
  _dstX(2) = x2
  _dstY(2) = y2
  _dstX(3) = x3
  _dstY(3) = y3
  _dirty = True
End Sub
```

Рабочей лошадкой этого класса является метод `ComputeWarp`, который вычисляет матрицу преобразования по исходным и конечным точкам. Сначала с помощью метода `ComputeSquareToQuad` вычисляется матрица преобразования единичного квадрата в конечный четырехугольник. Для этого нужно просто решить систему линейных уравнений. Затем

вызывается метод ComputeQuadToSquare, который вычисляет матрицу преобразования исходного четырехугольника в единичный квадрат, для чего, по существу, требуется обратить действия, выполняемые в методе ComputeSquareToQuad. Наконец, перемножив обе полученные матрицы, мы получим искомую матрицу преобразования исходной четверки в конечную (см. примеры 9.5 и 9.6).

Пример 9.5. Метод ComputeWarp и вспомогательные методы (версия на C#)

```
public void ComputeWarp()
{
  ComputeSquareToQuad( dstX[0], _dstY[0],
                       _dstX[1], _dstY[1],
                       _dstX[2], _dstY[2],
                       _dstX[3], _dstY[3],
                       _dstMat);
  ComputeQuadToSquare( srcX[0], _srcY[0],
                       _srcX[1], _srcY[1],
                       _srcX[2], _srcY[2],
                       _srcX[3], _srcY[3],
                       _srcMat);
  MultiplyMatrices(_srcMat, _dstMat, _warpMat);
  _dirty = false;
}

public void MultiplyMatrices(float[] srcMat, float[] dstMat, float[] resMat)
{
  // [ a b ]   [w x]   [(aw + by) (ax + bz)]
  // [ c d ] * [y z] = [(cw + dy) (cz + dz)]
  for (int row = 0; row < 4; row++)
  {
    int rowIndex = row * 4;
    for (int col = 0; col < 4; col++)
    {
      resMat[rowIndex + col] = (srcMat[rowIndex ] * dstMat[col ] +
                                srcMat[rowIndex + 1] * dstMat[col + 4] +
                                srcMat[rowIndex + 2] * dstMat[col + 8] +
                                srcMat[rowIndex + 3] * dstMat[col + 12]);
    }
  }
}

public void ComputeSquareToQuad( float x0,
                                 float y0,
                                 float x1,
                                 float y1,
                                 float x2,
                                 float y2,
                                 float x3,
                                 float y3,
                                 float[] mat)
{
```

```
  float dx1 = x1 - x2, dy1 = y1 - y2;
  float dx2 = x3 - x2, dy2 = y3 - y2;
  float sx = x0 - x1 + x2 - x3;
  float sy = y0 - y1 + y2 - y3;
  float g = (sx * dy2 - dx2 * sy) / (dx1 * dy2 - dx2 * dy1);
  float h = (dx1 * sy - sx * dy1) / (dx1 * dy2 - dx2 * dy1);
  float a = x1 - x0 + g * x1;
  float b = x3 - x0 + h * x3;
  float c = x0;
  float d = y1 - y0 + g * y1;
  float e = y3 - y0 + h * y3;
  float f = y0;
  mat[ 0] = a;   mat[ 1] = d;   mat[ 2] = 0;   mat[ 3] = g;
  mat[ 4] = b;   mat[ 5] = e;   mat[ 6] = 0;   mat[ 7] = h;
  mat[ 8] = 0;   mat[ 9] = 0;   mat[10] = 1;   mat[11] = 0;
  mat[12] = c;   mat[13] = f;   mat[14] = 0;   mat[15] = 1;
}

public void ComputeQuadToSquare( float x0,
                                 float y0,
                                 float x1,
                                 float y1,
                                 float x2,
                                 float y2,
                                 float x3,
                                 float y3,
                                 float[] mat)
{
  ComputeSquareToQuad(x0, y0, x1, y1, x2, y2, x3, y3, mat);

  // инвертировать с помощью сопряженной матрицы
  float a = mat[ 0], d = mat[ 1], /* игнорировать */ g = mat[ 3];
  float b = mat[ 4], e = mat[ 5], /* 3-ий столбец */ h = mat[ 7];
  /* игнорировать 3-ью строку */
  float c = mat[12], f = mat[13];
  float a1 = e - f * h;
  float b1 = c * h - b;
  float c1 = b * f - c * e;
  float d1 = f * g - d;
  float e1 = a - c * g;
  float f1 = c * d - a * f;
  float g1 = d * h - e * g;
  float h1 = b * g - a * h;
  float i1 = a * e - b * d;
  float idet = 1.0f / (a * a1 + b * d1 + c * g1);

  mat[ 0] = a1 * idet; mat[ 1] = d1 * idet; mat[ 2] = 0; mat[ 3] = g1 * idet;
  mat[ 4] = b1 * idet; mat[ 5] = e1 * idet; mat[ 6] = 0; mat[ 7] = h1 * idet;
  mat[ 8] = 0 ;        mat[ 9] = 0 ;        mat[10] = 1; mat[11] = 0 ;
  mat[12] = c1 * idet;  mat[13] = f1 * idet; mat[14] = 0; mat[15] = i1 * idet;
}

public float[] GetWarpMatrix()
```

```
{
  return _warpMat;
}
```

Пример 9.6. Метод ComputeWarp и вспомогательные методы (версия на VB)

```
Public Sub ComputeWarp()
    ComputeSquareToQuad(dstX(0), _dstY(0), _dstX(1), _dstY(1), _dstX(2), _
                        _dstY(2), _dstX(3), _dstY(3), _dstMat)
    ComputeQuadToSquare(srcX(0), _srcY(0), _srcX(1), _srcY(1), _srcX(2), _
                        _srcY(2), _srcX(3), _srcY(3), _srcMat)
    MultiplyMatrices(_srcMat, _dstMat, _warpMat)
    _dirty = False
End Sub

Public Sub MultiplyMatrices( ByVal srcMat() As Single, _
                             ByVal dstMat() As Single, _
                             ByVal resMat() As Single)
' [ a b ]   [w x]   [(aw + by) (ax + bz)]
' [ c d ] * [y z] = [(cw + dy) (cz + dz)]
  For row As Integer = 0 To 3
    Dim rowIndex As Integer = row * 4
    For col As Integer = 0 To 3
      resMat(rowIndex + col) = ( srcMat(rowIndex) * dstMat(col) + _
                                 srcMat(rowIndex + 1) * dstMat(col + 4)+ _
                                 srcMat(rowIndex + 2) * dstMat(col + 8) + _
                                 srcMat(rowIndex + 3) * dstMat(col + 12))
    Next col
  Next row
End Sub

Public Sub ComputeSquareToQuad( ByVal x0 As Single, ByVal y0 As Single, _
                                ByVal x1 As Single, ByVal y1 As Single, _
                                ByVal x2 As Single, ByVal y2 As Single, _
                                ByVal x3 As Single, ByVal y3 As Single, _
                                ByVal mat() As Single)

  Dim dx1 As Single = x1 - x2, dy1 As Single = y1 - y2
  Dim dx2 As Single = x3 - x2, dy2 As Single = y3 - y2
  Dim sx As Single = x0 - x1 + x2 - x3
  Dim sy As Single = y0 - y1 + y2 - y3
  Dim g As Single = (sx * dy2 - dx2 * sy) / (dx1 * dy2 - dx2 * dy1)
  Dim h As Single = (dx1 * sy - sx * dy1) / (dx1 * dy2 - dx2 * dy1)
  Dim a As Single = x1 - x0 + g * x1
  Dim b As Single = x3 - x0 + h * x3
  Dim c As Single = x0
  Dim d As Single = y1 - y0 + g * y1
  Dim e As Single = y3 - y0 + h * y3
  Dim f As Single = y0
  mat(0) = a
  mat(1) = d
  mat(2) = 0
  mat(3) = g
```

```
    mat(4) = b
    mat(5) = e
    mat(6) = 0
    mat(7) = h
    mat(8) = 0
    mat(9) = 0
    mat(10) = 1
    mat(11) = 0
    mat(12) = c
    mat(13) = f
    mat(14) = 0
    mat(15) = 1
  End Sub

  Public Sub ComputeQuadToSquare( ByVal x0 As Single, ByVal y0 As Single, _
                                  ByVal x1 As Single, ByVal y1 As Single, _
                                  ByVal x2 As Single, ByVal y2 As Single, _
                                  ByVal x3 As Single, ByVal y3 As Single, _
                                  ByVal mat() As Single)
    ComputeSquareToQuad(x0, y0, x1, y1, x2, y2, x3, y3, mat)

    ' инвертировать с помощью сопряженной матрицы
    ' игнорировать 3-ий столбец
    Dim a As Single = mat(0), d As Single = mat(1), g As Single = mat(3)
    Dim b As Single = mat(4), e As Single = mat(5), h As Single = mat(7)
    ' игнорировать 3-ью строку
    Dim c As Single = mat(12), f As Single = mat(13)
    Dim a1 As Single = e - f * h
    Dim b1 As Single = c * h - b
    Dim c1 As Single = b * f - c * e
    Dim d1 As Single = f * g - d
    Dim e1 As Single = a - c * g
    Dim f1 As Single = c * d - a * f
    Dim g1 As Single = d * h - e * g
    Dim h1 As Single = b * g - a * h
    Dim i1 As Single = a * e - b * d
    Dim idet As Single = 1.0f / (a * a1 + b * d1 + c * g1)
    mat(0) = a1 * idet
    mat(1) = d1 * idet
    mat(2) = 0
    mat(3) = g1 * idet
    mat(4) = b1 * idet
    mat(5) = e1 * idet
    mat(6) = 0
    mat(7) = h1 * idet
    mat(8) = 0
    mat(9) = 0
    mat(10) = 1
    mat(11) = 0
    mat(12) = c1 * idet
    mat(13) = f1 * idet
    mat(14) = 0
```

```
  mat(15) = i1 * idet
End Sub

Public Function GetWarpMatrix() As Single()
  Return _warpMat
End Function
```

Теперь, имея матрицу преобразования, мы можем получить образы любых точек, расположенных как внутри, так и вне четырехугольника, определяемого исходной четверкой. Для этого достаточно воспользоваться методом `Warp`, который умножает вектор `(srcX, srcY)` на матрицу преобразования, получая в результате вектор `(dstX, dstY)` (см. примеры 9.7 и 9.8). Статический вариант того же метода позволяет использовать альтернативную матрицу преобразования, если это почему-либо нужно.

Пример 9.7. Методы Warp (версия на C#)

```
public void Warp(float srcX, float srcY, ref float dstX, ref float dstY)
{
  if (_dirty)
    ComputeWarp();
  Warp(_warpMat, srcX, srcY, ref dstX, ref dstY);
}

public static void Warp( float[] mat, float srcX, float srcY, ref float dstX,
                         ref float dstY)
{
  float[] result = new float[4];
  float z = 0;
  result[0] = srcX * mat[0] + srcY*mat[4] + z*mat[ 8] + mat[12];
  result[1] = srcX * mat[1] + srcY*mat[5] + z*mat[ 9] + mat[13];
  result[2] = srcX * mat[2] + srcY*mat[6] + z*mat[10] + mat[14];
  result[3] = srcX * mat[3] + srcY*mat[7] + z*mat[11] + mat[15];
  dstX = result[0]/result[3];
  dstY = result[1]/result[3];
}
```

Пример 9.8. Методы Warp (версия на VB)

```
Public Sub Warp(ByVal srcX As Single, ByVal srcY As Single, _
                ByRef dstX As Single, ByRef dstY As Single)
    If _dirty Then
        ComputeWarp()
    End If

    Warp(_warpMat, srcX, srcY, dstX, dstY)
End Sub

Public Shared Sub Warp(ByVal mat() As Single, ByVal srcX As Single, _
                       ByVal srcY As Single, ByRef dstX As Single, _
                       ByRef dstY As Single)
    Dim result(3) As Single
    Dim z As Single = 0
```

```
        result(0) = srcX * mat(0) + srcY*mat(4) + z*mat(8) + mat(12)
        result(1) = srcX * mat(1) + srcY*mat(5) + z*mat(9) + mat(13)
        result(2) = srcX * mat(2) + srcY*mat(6) + z*mat(10) + mat(14)
        result(3) = srcX * mat(3) + srcY*mat(7) + z*mat(11) + mat(15)
        dstX = result(0)/result(3)
        dstY = result(1)/result(3)
    End Sub
```

Получив четыре пары исходных и конечных точек, компьютер может сопоставить каждому пикселу полученного от камеры изображения пиксель на плоском экране монитора. Теперь необходимо написать утилиту калибровки, которая позволит получить эти четыре пары точек.

Калибровка пера

У приложения WiimoteWhiteboard есть два графических интерфейса пользователя: главная панель управления и форма калибровки по четырем точкам. Назначение последней – получить четыре пары точек, необходимых для вычисления описанной выше матрицы проективного преобразования. Форма представляет собой полноэкранное окно, в котором в известных позициях вблизи от углов экрана расположены четыре крестика. Пользователь указывает ИК-пером на каждый крестик и включает светодиод; камера видит точку, программа сопоставляет положение этой точки с известными экранными координатами крестика, после чего переходит к следующему крестику. После регистрации всех четырех точек мы имеем достаточно информации для вычисления матрицы преобразования и можем приступать к эмуляции мыши.

Начнем с создания *калибровочной формы*. Добавьте в проект новую форму. Для этого щелкните правой кнопкой мыши по имени проекта и выберите из контекстного меню пункт Add→New Item. Затем из списка шаблонов выберите *Windows Form* и назовите форму *CalibrationForm.cs* или *CalibrationForm.vb*.

Необходимо установить свойства формы так, чтобы она занимала весь экран. Для этого присвойте свойству `FormBorderStyle` значение `None`, а свойству `StartPosition` – значение `Manual`. Позже в тексте программы мы вручную установим позицию и размер окна так, чтобы оно занимало весь экран.

Мы хотим рисовать на поверхности этой формы; один из способов решить эту задачу – воспользоваться объектом `PictureBox`. Перетащите из инструментария элемент *PictureBox* и нарисуйте прямоугольник на поверхности формы. Назовите этот элемент `pbCalibrate`. Его размер и положение сейчас не важны, позже мы установим их программно.

Теперь зададим обработчик события `KeyDown`. Для этого щелкните по значку молнии в окне свойств Properties window, а затем дважды щелкните по строке KeyDown, как показано на рис. 9.4.

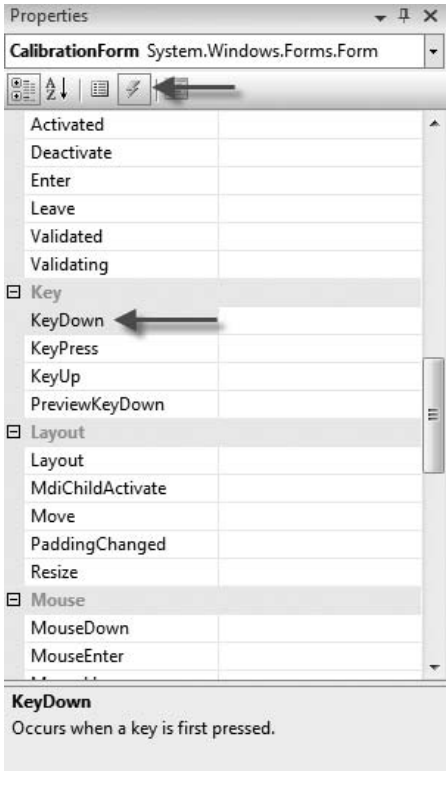

Рис. 9.4. *Событие KeyDown*

При этом будет создан пустой метод, который мы заполним, как показано в примерах 9.9 и 9.10. Смысл этого обработчика в том, чтобы закрыть окно до завершения калибровки при нажатии клавиши Esc.

Пример 9.9. Обработчик события KeyDown (версия на C#)

```
private void CalibrationForm_KeyDown(object sender, KeyEventArgs e)
{
  if (e.KeyCode == Keys.Escape)
    this.Close();
}
```

Пример 9.10. Обработчик события KeyDown (версия на VB)

```
Private Sub CalibrationForm_KeyDown(ByVal sender As Object, _
                                    ByVal e As KeyEventArgs) Handles MyBase.KeyDown
    If e.KeyCode = Keys.Escape Then
        Me.Close()
    End If
End Sub
```

Далее нужно добавить переменные-члены, содержащие ссылки на растровое изображение, которое мы рисуем, и соответствующий ему графический контекст. Затем напишем конструктор формы. Он должен выполнять несколько действий. Сначала он получает от системы размеры экрана и распахивает форму на весь экран. Затем он инициализирует растр такого же размера, как экран, инициализирует графический контекст и устанавливает размеры ранее созданного элемента `PictureBox` также во весь экран. Наконец, он задает начальный белый цвет растра и активизирует обновление элемента `PictureBox`, который будет рисовать на экране (примеры 9.11 и 9.12).

Пример 9.11. Переменные-члены и конструктор формы CalibrationForm (версия на C#)

```
using System.Drawing;
using System.Drawing.Imaging;
using System.Windows.Forms;

public partial class CalibrationForm : Form
{
  private Bitmap _bmpCalibration;
  private Graphics _gfxCalibration;

  public CalibrationForm()
  {
    InitializeComponent();
    Rectangle rect = Screen.GetWorkingArea(this);
    this.Size = new Size(rect.Width, rect.Height);
    this.Text = "Calibration - Working area:" +
                Screen.GetWorkingArea(this).ToString() + " || Real area: " +
               Screen.GetBounds(this).ToString();

    _bmpCalibration = new Bitmap(rect.Width, rect.Height,
                                 PixelFormat.Format24bppRgb);
    _gfxCalibration = Graphics.FromImage(_bmpCalibration);
    pbCalibrate.Left = 0;
    pbCalibrate.Top = 0;
    pbCalibrate.Size = new Size(rect.Width, rect.Height);
    pbCalibrate.Image = _bmpCalibration;

    _gfxCalibration.Clear(Color.White);
  }
```

Пример 9.12. Переменные-члены и конструктор формы CalibrationForm (версия на VB)

```
Imports System.Drawing
Imports System.Drawing.Imaging
Imports System.Windows.Forms

Namespace WiimoteWhiteboard
    Partial Public Class CalibrationForm
        Inherits Form
        Private _bmpCalibration As Bitmap
```

```
Private _gfxCalibration As Graphics

Public Sub New()
    InitializeComponent()

    Dim rect As Rectangle = Screen.GetWorkingArea(Me)

    Me.Size = New Size(rect.Width, rect.Height)
    Me.Text = "Calibration - Working area:" & _
              Screen.GetWorkingArea(Me).ToString() & _
              " || Real area: " & Screen.GetBounds(Me).ToString()

    _bmpCalibration = New Bitmap(rect.Width, rect.Height, _
                                 PixelFormat.Format24bppRgb)
    _gfxCalibration = Graphics.FromImage(_bmpCalibration)
    pbCalibrate.Left = 0
    pbCalibrate.Top = 0
    pbCalibrate.Size = New Size(rect.Width, rect.Height)
    pbCalibrate.Image = _bmpCalibration

    _gfxCalibration.Clear(Color.White)
End Sub
```

В примерах 9.13 и 9.14 показана оставшаяся часть класса `CalibrationForm` **– процедура, которая очищает экран, рисует один крестик и выводит его на экран. Этот метод главная программа будет вызывать для выполнения каждого из четырех шагов калибровки.**

Пример 9.13. Оставшаяся часть класса CalibrationForm (версия на C#)

```
public void ShowCalibration(int x, int y, int size, Pen p)
{
  _gfxCalibration.Clear(Color.White);

  // нарисовать крестик
  _gfxCalibration.DrawEllipse(p, x - size / 2, y - size / 2, size, size);
  _gfxCalibration.DrawLine(p, x-size, y, x+size, y);
  _gfxCalibration.DrawLine(p, x, y-size, x, y+size);

  BeginInvoke((MethodInvoker)delegate() { pbCalibrate.Image =
  _bmpCalibration; });
}
```

Пример 9.14. Оставшаяся часть класса CalibrationForm (версия на VB)

```
Public Sub ShowCalibration(ByVal x As Integer, ByVal y As Integer,_
                           ByVal size As Integer, ByVal p As Pen)
    _gfxCalibration.Clear(Color.White)

    ' нарисовать крестик
    _gfxCalibration.DrawEllipse(p, x - size \ 2, y - size \ 2, size, size)
    _gfxCalibration.DrawLine(p, x-size, y, x+size, y)
    _gfxCalibration.DrawLine(p, x, y-size, x, y+size)

    BeginInvoke(CType(AddressOf SetImage, MethodInvoker))
End Sub
```

```
Private Sub SetImage()
    pbCalibrate.Image = _bmpCalibration
End Sub
```

Собираем все вместе

Теперь, когда у нас есть оба компонента, `Warper` и `CalibrationForm`, необходимо написать код главной формы, которая будет координировать их работу: показывать графическую панель управления, выполнять процедуру калибровки, инициировать вычисление матрицы преобразования и эмулировать мышь. На первый взгляд, довольно много, но реализуется очень просто.

Для начала добавим в пользовательский интерфейс визуальные компоненты. Двойным щелчком по форме *Form1.cs/vb* откройте ее конструктор. Для индикации уровня заряда батарейки создайте элемент *GroupBox* с меткой «Wiimote Battery». В него поместите элемент *ProgressBar* с именем «pbBattery» и метку *Label* с именем «lblBattery». Они будут давать графическое и текстовое представление заряда батарейки.

Под элементом *GroupBox* расположите еще одну метку с именем «lblIRVisible». Это диагностический элемент, который говорит о том, видит ли камера ИК-перо. Он поможет нам отлаживать неисправности, связанные с видимостью изображения на экране.

Добавьте еще одну метку «lblTrackingUtil». И это средство диагностики, позволяющее судить о качестве позиционирования Wiimote. В нем отображается, какой процент получаемого от камеры изображения используется для рисования следа на экране. Это грубая оценка качества слежения.

Под этой меткой разместите флажок *CheckBox*, который будет включать и выключать управление курсором мыши; назовите его «cbCursorControl». Он поможет подавить беспорядочное метание курсора во время позиционирования Wiimote и тестирования камеры.

И наконец, добавьте большую кнопку *Button* для запуска процедуры калибровки, присвойте ей имя «btnCalibrate». По завершении работы интерфейс должен выглядеть, как показано на рис. 9.5.

Теперь напишем код. Снова щелкните правой кнопкой мыши по файлу *Form1.cs/vb* и выберите из контекстного меню команду `View Code`. Нам потребуется завести в классе формы несколько переменных-членов, показанных в примерах 9.15 и 9.16: описатель Wiimote, кое-какие служебные переменные, ссылку на форму калибровки и объект *Warper*.

Но прежде добавьте ссылку на библиотеку *WiimoteLib*. Скопируйте файл *WiimoteLib.dll* из ее дистрибутива в локальный каталог проекта, чтобы его проще было найти, а затем добавьте на него ссылку.

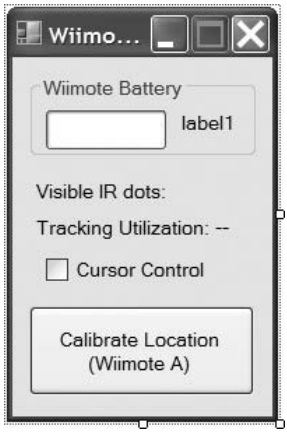

Рис. 9.5. *Графический интерфейс главной формы*

Пример 9.15. Переменные-члены класса Form1 (версия на C#)

```
using System;
using System.Drawing;
using System.Windows.Forms;
using System.Runtime.InteropServices;
using System.IO;
using WiimoteLib;

namespace WiimoteWhiteboard
{
  public partial class Form1 : Form
  {
    private const string CalibrationFilename = "calibration.dat";

    // экземпляр wii remote
    private Wiimote _wm = new Wiimote();

    // нужен для возбуждения событий
    private WiimoteState _lastWiiState = new WiimoteState();

    private bool _cursorControl;
    private Size _screenSize;

    private int _calibrationState;
    // как далеко крестики должны отстоять от края экрана
    private float _calibrationMargin = .1f;

    // количество сохраненных точек
    int _smoothingCount = 0;

    // по скольким точкам усреднять (1-10)
    int _smoothingPoints = 5;

    // прошлые координаты для сглаживания
```

```
    float[] _smoothingX = new float[10];
    float[] _smoothingY = new float[10];

    private CalibrationForm cf;

    private Warper _warper = new Warper();
    private float[] _srcX = new float[4];
    private float[] _srcY = new float[4];
    private float[] _dstX = new float[4];
    private float[] _dstY = new float[4];

    public Form1()
    {
      InitializeComponent();
    }
  }
}
```

Пример 9.16. Переменные-члены класса Form1 (версия на VB)

```
Imports Microsoft.VisualBasic
Imports System
Imports System.Drawing
Imports System.Windows.Forms
Imports System.Runtime.InteropServices
Imports System.IO
Imports WiimoteLib

Namespace WiimoteWhiteboard
  Partial Public Class Form1
    Inherits Form
    Private Const CalibrationFilename As String = "calibration.dat"

    ' экземпляр wii remote
    Private _wm As New Wiimote()

    // нужен для возбуждения событий
    Private _lastWiiState As New WiimoteState()

    Private _cursorControl As Boolean
    Private _screenSize As Size

    Private _calibrationState As Integer

    ' как далеко крестики должны отстоять от края экрана
    Private _calibrationMargin As Single =.1f

    ' количество сохраненных точек
    Private _smoothingCount As Integer = 0

    ' по скольким точкам усреднять (1-10)
    Private _smoothingPoints As Integer = 5

    ' прошлые координаты для сглаживания
    Private _smoothingX(9) As Single
    Private _smoothingY(9) As Single
```

```
    Private cf As CalibrationForm

    Private _warper As New Warper()
    Private _srcX(3) As Single
    Private _srcY(3) As Single
    Private _dstX(3) As Single
    Private _dstY(3) As Single

    Public Sub New()
      InitializeComponent()
    End Sub
  End Class
End Namespace
```

Мы хотим эмулировать поведение мыши и клавиатуры. Проще всего сделать это, программно генерируя события, которые возникали бы, если бы пользователь просто двигал мышь или печатал на клавиатуре. Для этого следует искусственно сконструировать эти события, поместив в них необходимые данные, и отправить операционной системе. К сожалению, .NET не поддерживает все, что нам нужно, поэтому придется обратиться к Win32 API с помощью служб поддержки совместимости и механизма P/Invoke. В самом начале класса поместим директиву `using System.Runtime.InteropServices`, которая даст доступ к неуправляемым библиотекам Win32 для эмуляции событий ввода. Наибольший интерес для нас представляет функция `SendInput`, отправляющая события операционной системе. Ей необходимо передать правильно подготовленные структуры данных `INPUT`, в каждую из которых вложена еще и структура `MOUSEINPUT`. Кроме того, мы определим несколько констант, описывающих отдельные поля информации о событии. Наконец, мы импортируем функцию `keybd_event`, которая упрощает генерацию событий клавиатуры (см. примеры 9.17 и 9.18).

Пример 9.17. Константы, структуры и импорт функций Win32 API (версия на C#)

```
//объявляем константы для сообщений мыши
public const int INPUT_MOUSE = 0;
public const int MOUSEEVENTF_MOVE = 0x01;
public const int MOUSEEVENTF_LEFTDOWN = 0x02;
public const int MOUSEEVENTF_LEFTUP = 0x04;
public const int MOUSEEVENTF_RIGHTDOWN = 0x08;
public const int MOUSEEVENTF_RIGHTUP = 0x10;
public const int MOUSEEVENTF_ABSOLUTE = 0x8000;

//объявляем константы для скан-кодов клавиш
public const byte VK_LEFT = 0x25;
public const byte VK_UP = 0x26;
public const byte VK_RIGHT = 0x27;
public const byte VK_DOWN = 0x28;
public const int KEYEVENTF_KEYUP = 0x02;

public struct MOUSEINPUT //24 байта
{
```

```
  public int dx; //4
  public int dy; //4
  public uint mouseData; //4
  public uint dwFlags; //4
  public uint time; //4
  public IntPtr dwExtraInfo;//4
}

public struct INPUT //28 байтов
{
  public int type; // 4 байта
  public MOUSEINPUT mi; //24 байта
}

private const int INPUT_SIZE = 28;
private INPUT[] _buffer = new INPUT[2];

//для возбуждения событий мыши
[DllImport("user32.dll", SetLastError = true)]
static extern uint SendInput(uint nInputs, INPUT[] pInputs, int cbSize);

//импортируем функцию keybd_event из user32.dll
[DllImport("user32.dll", SetLastError = true)]
public static extern void keybd_event(byte bVk, byte bScan, long dwFlags,
                                      long dwExtraInfo);
```

Пример 9.18. Константы, структуры и импорт функций Win32 API (версия на VB)

```
' объявляем константы для сообщений мыши
Public Const INPUT_MOUSE As Integer = 0
Public Const MOUSEEVENTF_MOVE As Integer = &H01
Public Const MOUSEEVENTF_LEFTDOWN As Integer = &H02
Public Const MOUSEEVENTF_LEFTUP As Integer = &H04
Public Const MOUSEEVENTF_RIGHTDOWN As Integer = &H08
Public Const MOUSEEVENTF_RIGHTUP As Integer = &H10
Public Const MOUSEEVENTF_ABSOLUTE As Integer = &H8000

' объявляем константы для скан-кодов клавиш
Public Const VK_LEFT As Byte = &H25
Public Const VK_UP As Byte = &H26
Public Const VK_RIGHT As Byte = &H27
Public Const VK_DOWN As Byte = &H28
Public Const KEYEVENTF_KEYUP As Integer = &H02

Public Structure MOUSEINPUT '24 байта
  Public dx As Integer '4
  Public dy As Integer '4
  Public mouseData As UInteger '4
  Public dwFlags As UInteger '4
  Public time As UInteger '4
  Public dwExtraInfo As IntPtr '4
End Structure

Public Structure INPUT '28 байтов
```

```
    Public type As Integer ' 4 байта
    Public mi As MOUSEINPUT '24 байта
End Structure

Private Const INPUT_SIZE As Integer = 28
' для возбуждения событий мыши
<DllImport("user32.dll", SetLastError := True)> _
Shared Function SendInput(ByVal nInputs As UInteger, _
                          ByVal pInputs() As INPUT, _
                          ByVal cbSize As Integer) As UInteger
End Function

' импортируем функцию keybd_event из user32.dll
<DllImport("user32.dll", SetLastError := True)> _
Public Shared Sub keybd_event(ByVal bVk As Byte, ByVal bScan As Byte, _
                              ByVal dwFlags As Long, ByVal dwExtraInfo As Long)
End Sub

Private _buffer(1) As INPUT

' делегат для обновления пользовательского интерфейса
' в создавшем его потоке
Private Delegate Sub UpdateUIDelegate(ByVal args As WiimoteState)

' делегат для обновления информации слежения
Private Delegate Sub UpdateTrackingUtilizationDelegate(ByVal utilStatus _
                                                       As String)
```

Объявив переменные-члены и импортировав необходимые процедуры, можно приступить к написанию кода главной программы. После загрузки формы мы можем инициализировать некоторые переменные, в частности размеры экрана и соединение с Wiimote. Если установить соединение не удастся, мы выведем сообщение об ошибке и закроем форму. В противном случае установим обработчик события, который будет вызываться всякий раз при поступлении данных от Wiimote, затем загрузим файл калибровки, если таковой существует. Полезно иметь в виду, что данные от Wiimote поступают примерно 100 раз в секунду.

Далее наделим функциональностью элементы пользовательского интерфейса. При закрытии формы надо не забыть разорвать соединение с Wiimote. После нажатия кнопки Calibrate следует открыть форму и запустить процедуру калибровки. Наконец, переменную, активирующую управление курсором мыши, мы привязываем к состоянию флажка.

Для начала откройте в Visual Studio окно свойств событий и добавьте обработчики по умолчанию для следующих событий:

- `Form1_Load`
- `Form1_Closed`
- `btnCalibrate_Click`
- `cbCursorControl_CheckChanged`

Чтобы это сделать, щелкните по соответствующему элементу управления в конструкторе, а затем дважды щелкните по интересующей нас строке в списке событий. Будет создан пустой обработчик. Полная реализация всех обработчиков приведена в примерах 9.19 и 9.20.

Пример 9.19. Обработчики событий Form1 (версия на C#)

```
private void Form1_Load(object sender, EventArgs e)
{
  _screenSize.Width = Screen.GetBounds(this).Width;
  _screenSize.Height= Screen.GetBounds(this).Height;
  try
  {
    //установить соединение с wii remote
    _wm.Connect();

    //указать, какие возможности пульта нас интересуют
    //полный перечень см. в документации по Wiimote.InputReport
    _wm.SetReportType(InputReport.IRAccel, true);

    //настроить светодиоды wiiremote
    _wm.SetLEDs(true, false, false, false);
  }
  catch (Exception x)
  {
    MessageBox.Show("Исключение: " + x.Message);
    this.Close();
  }

  //добавить обработчики изменений состояния wiiremote
  //вызываются при каждом поступлении данных -
  //обычно 100 раз в сек, если акселерометр включен
  _wm.WiimoteChanged += wm_WiimoteChanged;

  LoadCalibrationData();
}

private void Form1_FormClosed(object sender, FormClosedEventArgs e)
{
  //разорвать соединение с wiimote
  _wm.Disconnect();
}

private void btnCalibrate_Click(object sender, EventArgs e)
{
  if(cf == null || cf.IsDisposed)
           cf = new CalibrationForm();
  cf.Show();
  _cursorControl = false;
  _calibrationState = 1;
  DoCalibration();
}

private void cbCursorControl_CheckedChanged(object sender, EventArgs e)
{
```

```
  _cursorControl = cbCursorControl.Checked;
}
```

Пример 9.20. Обработчики событий Form1 (версия на VB)

```
Private Sub Form1_Load(ByVal sender As Object, ByVal e As EventArgs) _
        Handles MyBase.Load
        _screenSize.Width = Screen.GetBounds(Me).Width
        _screenSize.Height= Screen.GetBounds(Me).Height

        Try
                ' установить соединение с wii remote
                _wm.Connect()

                'указать, какие возможности пульта нас интересуют
                'полный перечень см. в документации по Wiimote.InputReport
                _wm.SetReportType(InputReport.IRAccel, True)

                'настроить светодиоды wiiremote
                _wm.SetLEDs(True, False, False, False)
        Catch x As Exception
                MessageBox.Show("Исключение: " & x.Message)
                Me.Close()
        End Try

        'добавить обработчики изменений состояния wiiremote
        'вызываются при каждом поступлении данных -
        'обычно 100 раз в сек, если акселерометр включен
        AddHandler _wm.WiimoteChanged, AddressOf wm_WiimoteChanged

        LoadCalibrationData()
End Sub

Private Sub Form1_FormClosed(ByVal sender As Object, _
                             ByVal e As FormClosedEventArgs) _
                             Handles MyBase.FormClosed
        'разорвать соединение с wiimote
        _wm.Disconnect()
End Sub

Private Sub btnCalibrate_Click(ByVal sender As Object, ByVal e As EventArgs) _
                              Handles btnCalibrate.Click
        If cf Is Nothing OrElse cf.IsDisposed Then
                cf = New CalibrationForm()
        End If

        cf.Show()
        _cursorControl = False
        _calibrationState = 1
        DoCalibration()
End Sub

Private Sub cbCursorControl_CheckedChanged(ByVal sender As Object, _
                                           ByVal e As EventArgs) _
                                     Handles cbCursorControl.CheckedChanged
        _cursorControl = cbCursorControl.Checked
End Sub
```

Процедура калибровки представляет собой простой конечный автомат, который переходит от одной точки к другой и по завершении вычисляет матрицу преобразования. Поэтому в методе `DoCalibration` имеется предложение `switch` с пятью ветками. В первых четырех ветках мы говорим, где на экране рисовать крестик. Переменная `_calibrationMargin` определяет, насколько далеко крестик должен отстоять от края экрана. Позиции крестиков – это конечная четверка, необходимая для вычисления матрицы проективного преобразования. В последней ветке вычисляется сама матрица, закрывается форма калибровки и данные калибровки сохраняются в файле (см. примеры 9.21 и 9.22).

Пример 9.21. Метод DoCalibration (версия на C#)

```
public void DoCalibration()
{
  if (cf == null)
    return;
  int x;
  int y;
  int size = 25;
  Pen p = new Pen(Color.Red);

  switch (_calibrationState)
  {
    case 1:
      x = (int)(_screenSize.Width * _calibrationMargin);
      y = (int)(_screenSize.Height * _calibrationMargin);
      cf.ShowCalibration(x, y, size, p);
      _dstX[_calibrationState - 1] = x;
      _dstY[_calibrationState - 1] = y;
      break;
    case 2:
      x = _screenSize.Width - (int)(_screenSize.Width * _calibrationMargin);
      y = (int)(_screenSize.Height * _calibrationMargin);
      cf.ShowCalibration(x, y, size, p);
      _dstX[_calibrationState - 1] = x;
      _dstY[_calibrationState - 1] = y;
      break;
    case 3:
      x = (int)(_screenSize.Width * _calibrationMargin);
      y = _screenSize.Height -(int)(_screenSize.Height * _calibrationMargin);
      cf.ShowCalibration(x, y, size, p);
      _dstX[_calibrationState - 1] = x;
      _dstY[_calibrationState - 1] = y;
      break;
    case 4:
      x = _screenSize.Width - (int)(_screenSize.Width * _calibrationMargin);
      y = _screenSize.Height -(int)(_screenSize.Height * _calibrationMargin);
      cf.ShowCalibration(x, y, size, p);
      _dstX[_calibrationState - 1] = x;
      _dstY[_calibrationState - 1] = y;
```

```
      break;
    case 5:
      //вычислить матрицу преобразования
      _warper.SetDestination( _dstX[0], _dstY[0], _dstX[1], _dstY[1],
                              _dstX[2], _dstY[2], _dstX[3], _dstY[3]);
      _warper.SetSource(_srcX[0], _srcY[0], _srcX[1], _srcY[1], _srcX[2],
                        _srcY[2], _srcX[3], _srcY[3]);
      _warper.ComputeWarp();
      _calibrationState = 0;
      _cursorControl = true;
      BeginInvoke(( MethodInvoker)delegate() {
                  cf.Close();
                  cbCursorControl.Checked = _cursorControl;
      });
      SaveCalibrationData();
      UpdateTrackingUtilization();
      break;
    default:
      break;
  }
}
```

Пример 9.22. Метод DoCalibration (версия на VB)

```
Public Sub DoCalibration()
  If cf Is Nothing Then
    Return
  End If

  Dim x As Integer
  Dim y As Integer
  Dim size As Integer = 25
  Dim p As New Pen(Color.Red)

  Select Case _calibrationState
    Case 1
      x = CInt(Fix(_screenSize.Width * _calibrationMargin))
      y = CInt(Fix(_screenSize.Height * _calibrationMargin))
      cf.ShowCalibration(x, y, size, p)
      _dstX(_calibrationState - 1) = x
      _dstY(_calibrationState - 1) = y
    Case 2
      x = _screenSize.Width - CInt(Fix(_screenSize.Width * _
                                       _calibrationMargin))
      y = CInt(Fix(_screenSize.Height * _calibrationMargin))
      cf.ShowCalibration(x, y, size, p)
      _dstX(_calibrationState - 1) = x
      _dstY(_calibrationState - 1) = y
    Case 3
      x = CInt(Fix(_screenSize.Width * _calibrationMargin))
      y = _screenSize.Height -CInt(Fix(_screenSize.Height * _
                                       _calibrationMargin))
      cf.ShowCalibration(x, y, size, p)
```

```
        _dstX(_calibrationState - 1) = x
        _dstY(_calibrationState - 1) = y
      Case 4
        x = _screenSize.Width - CInt(Fix(_screenSize.Width * _
                                         _calibrationMargin))
        y = _screenSize.Height - CInt(Fix(_screenSize.Height * _
                                          _calibrationMargin))
        cf.ShowCalibration(x, y, size, p)
        _dstX(_calibrationState - 1) = x
        _dstY(_calibrationState - 1) = y
      Case 5
        ' вычислить матрицу преобразования
        _warper.SetDestination(_dstX(0), _dstY(0), _dstX(1), _dstY(1), _
                               _dstX(2), _dstY(2), _dstX(3), _dstY(3))
        _warper.SetSource(_srcX(0), _srcY(0), _srcX(1), _srcY(1), _srcX(2), _
                          _srcY(2), _srcX(3), _srcY(3))
        _warper.ComputeWarp()
        _calibrationState = 0
        _cursorControl = True
        BeginInvoke(CType(AddressOf AnonymousMethod2, MethodInvoker))
        SaveCalibrationData()
        UpdateTrackingUtilization()
      Case Else
    End Select
End Sub

Private Sub CloseForm()
    cf.Close()
    cbCursorControl.Checked = _cursorControl
End Sub
```

Рабочей лошадкой всей программы является обработчик события, в котором разбираются данные от пульта Wiimote. Он вызывается 100 раз в секунду по мере поступления новых данных. В зависимости от текущего состояния программы данные обрабатываются по-разному. Для эмуляции мыши мы должны установить соответствие между поведением ИК-луча и событиями мыши. Проще всего считать, что кнопка на ИК-пере – эквивалент кнопки мыши. При нажатии кнопки на пере зажигается светодиод, камера находит новую точку, и мы генерируем событие нажатия на кнопку мыши в соответствующей ей точке экрана. При отпускании кнопки на пере точка исчезает, это соответствует событию отпускания кнопки мыши. Если точка перемещается, значит, пользователь водит пером с нажатой кнопкой, – мы интерпретируем такое поведение, как буксировку мышью.

Анализ данных от ИК-камеры

Пульт Wiimote автоматически нумерует до четырех видимых точек в порядке их появления. В настоящей программе предполагается, что единственной видимой точкой будет след, оставляемый пером. В результате мы всегда смотрим на точку с номером 1. Если эта точка ви-

дима, значит, она либо только что появилась, либо это изменившаяся позиция точки, которую мы уже видели раньше. В любом случае программа первым делом вычисляет соответствующую точку на экране, пользуясь построенной в ходе калибровки матрицей преобразования.

Чтобы понять, имеем ли мы дело с новой точкой, нужно сравнить ее с предшествующим состоянием Wiimote, которое хранится в переменной `_lastWiiState`. Если раньше мы эту точку не видели, значит, она только что появилась. Если управление курсором включено, то в этот момент мы эмулируем нажатие кнопки мыши. На самом деле следует сгенерировать два события: перемещение курсора к той точке, где находится перо, и нажатие кнопки. Если этого не сделать, то событие будет интерпретировано, как очень быстрая буксировка из предыдущего места нахождения курсора в новое. Чтобы создать оба события, мы используем переменную `_buffer`, которая достаточно велика, чтобы вместить две структуры `INPUT`. Обе структуры определены как события мыши. В первом событии мы перемещаем курсор в точку на экране, соответствующую текущему положению пера. Microsoft Windows ожидает, что координаты мыши приведены к диапазону 0-65535. Второе событие эмулирует нажатие левой кнопки мыши в точке, где находится курсор спустя 10 мсек. Подготовленные структуры передаются функции `SendInput`, которая помещает их в очередь событий таким образом, что никакая программа не отличит от их обычных событий мыши.

Если программа в данный момент занята калибровкой, то мы сохраняем координаты ИК-точки в массиве, представляющем «исходную четверку» (точки в системе координат камеры, соответствующие крестикам на экране), и увеличиваем номер состояния калибровки. Если точка не новая, то порождается событие перемещения мыши. Если точка сейчас не видна, но была видна в последний раз, значит, она исчезла, и мы порождаем событие отпускания левой кнопки мыши. Некоторые программы, особенно ориентированные на планшетные перья, ожидают, что за событием отпускания кнопки последует событие перемещения, иначе их интерфейс работает некорректно. Чтобы успокоить такие программы, мы генерируем событие перемещения мыши на нулевое расстояние.

Во время буксировки мы сглаживаем поступающие от Wiimote данные. Сохраняя положения последних пяти полученных от Wiimote точек и затем усредняя по ним, мы можем построить гораздо более плавную траекторию. В момент, когда виртуальная кнопка мыши отпускается, мы очищаем сохраненные данные, чтобы потом начать их накопление снова.

Наконец, мы запоминаем, была ли видна точка в этот раз, чтобы сравнить с новым состоянием при следующем вызове (см. примеры 9.23 и 9.24).

Пример 9.23. Первая половина обработчика события WiimoteChanged (версия на C#)

```
void wm_WiimoteChanged(object sender, WiimoteChangedEventArgs e)
{
  //получить состояние wiimote
  WiimoteState ws = e.WiimoteState;
  if (ws.IRState.IRSensors[0].Found)
  {
    int x = ws.IRState.IRSensors[0].RawPosition.X;
    int y = ws.IRState.IRSensors[0].RawPosition.Y;
    float warpedX = x;
    float warpedY = y;
    _warper.Warp(x, y, ref warpedX, ref warpedY);
    if (!_lastWiiState.IRState.IRSensors[0].Found)//кнопка нажата
    {
       _lastWiiState.IRState.IRSensors[0].Found =
            ws.IRState.IRSensors[0].Found;
        if (_cursorControl)
        {
          _buffer[0].type = INPUT_MOUSE;
          _buffer[0].mi.dx = (int)(warpedX * 65535.0f / _screenSize.Width);
          _buffer[0].mi.dy = (int)(warpedY * 65535.0f / _screenSize.Height);
          _buffer[0].mi.mouseData = 0;
          _buffer[0].mi.dwFlags = MOUSEEVENTF_ABSOLUTE | MOUSEEVENTF_MOVE;
          _buffer[0].mi.time = 0;
          _buffer[0].mi.dwExtraInfo = (IntPtr)0;
          _buffer[1].type = INPUT_MOUSE;
          _buffer[1].mi.dx = 0;
          _buffer[1].mi.dy = 0;
          _buffer[1].mi.mouseData = 0;
          _buffer[1].mi.dwFlags = MOUSEEVENTF_LEFTDOWN;
          _buffer[1].mi.time = 10;
          _buffer[1].mi.dwExtraInfo = (IntPtr)0;

          SendInput(2, _buffer, INPUT_SIZE);
        }//управление курсором

        switch (_calibrationState)
        {
          case 1:
            _srcX[_calibrationState - 1] = x;
            _srcY[_calibrationState - 1] = y;
            _calibrationState = 2;
            DoCalibration();
            break;
          case 2:
            _srcX[_calibrationState - 1] = x;
            _srcY[_calibrationState - 1] = y;
            _calibrationState = 3;
            DoCalibration();
            break;
```

```
      case 3:
        _srcX[_calibrationState - 1] = x;
        _srcY[_calibrationState - 1] = y;
        _calibrationState = 4;
        DoCalibration();
        break;
      case 4:
        _srcX[_calibrationState - 1] = x;
        _srcY[_calibrationState - 1] = y;
        _calibrationState = 5;
        DoCalibration();
        break;
      default:
        break;
    }//калибровка
  }//кнопка нажата
  else
  {
    if (_cursorControl)//буксировка
    {
      float sumX = warpedX, sumY = warpedY; //инициализировать сумму
      _smoothingCount += 1; //количество хранимых значений

      if (_smoothingCount > _smoothingPoints)
        _smoothingCount = _smoothingPoints;

      for (int i = 0; i < _smoothingPoints - 1; i++)
      {
        //сдвинуть хранимые точки
        _smoothingX[i] = _smoothingX[i + 1];
        _smoothingY[i] = _smoothingY[i + 1];

        sumX += _smoothingX[i]; //увеличить сумму
        sumY += _smoothingY[i];
      }

      //добавить новую точку в последнюю позицию
      _smoothingX[_smoothingPoints - 1] = warpedX;
      _smoothingY[_smoothingPoints - 1] = warpedY;

      //вычислить средние
      float smoothWarpedX = sumX / (_smoothingCount);
      float smoothWarpedY = sumY / (_smoothingCount);

      //буксировка в точку с усредненными координатами
      _buffer[0].type = INPUT_MOUSE;
      _buffer[0].mi.dx = (int)(smoothWarpedX * 65535.0f /
                          _screenSize.Width);
      _buffer[0].mi.dy = (int)(smoothWarpedY * 65535.0f /
                          _screenSize.Height);
      _buffer[0].mi.mouseData = 0;
      _buffer[0].mi.dwFlags = MOUSEEVENTF_ABSOLUTE | MOUSEEVENTF_MOVE;
      _buffer[0].mi.time = 0;
      _buffer[0].mi.dwExtraInfo = (IntPtr)0;
```

```
            SendInput(1, _buffer, INPUT_SIZE);
        }
    }
}//ИК-точка видна
else
{
        if (_lastWiiState.IRState.IRSensors[0].Found)//кнопка отпущена
        {
          if (_cursorControl)
          {
            _buffer[0].type = INPUT_MOUSE;
            _buffer[0].mi.dx = 0;
            _buffer[0].mi.dy = 0;
            _buffer[0].mi.mouseData = 0;
            _buffer[0].mi.dwFlags = MOUSEEVENTF_LEFTUP;
            _buffer[0].mi.time = 0;
            _buffer[0].mi.dwExtraInfo = (IntPtr)0;
            _buffer[1].type = INPUT_MOUSE;
            _buffer[1].mi.dx = 0;
            _buffer[1].mi.dy = 0;
            _buffer[1].mi.mouseData = 0;
            _buffer[1].mi.dwFlags = MOUSEEVENTF_MOVE;
            _buffer[1].mi.time = 0;
            _buffer[1].mi.dwExtraInfo = (IntPtr)0;
            SendInput(2, _buffer, INPUT_SIZE);

            //очистить сглаженные данные при отпускании кнопки
            ResetCursorSmoothing();
          }
        }//ИК-точка пропала
}

_lastWiiState.IRState.IRSensors[0].Found =
          ws.IRState.IRSensors[0].Found;
```

Пример 9.24. Первая половина обработчика события WiimoteChanged (версия на VB)

```
Private Sub wm_WiimoteChanged(ByVal sender As Object, _
                              ByVal e As WiimoteChangedEventArgs)
  'получить состояние wiimote
  Dim ws As WiimoteState = e.WiimoteState

  If ws.IRState.IRSensors(0).Found Then
    Dim x As Integer = ws.IRState.IRSensors(0).RawPosition.X
    Dim y As Integer = ws.IRState.IRSensors(0).RawPosition.Y
    Dim warpedX As Single = x
    Dim warpedY As Single = y

    _warper.Warp(x, y, warpedX, warpedY)

    If (Not _lastWiiState.IRState.IRSensors(0).Found) Then 'кнопка нажата
        _lastWiiState.IRState.IRSensors(0).Found = _
            ws.IRState.IRSensors(0).Found
```

```
  If _cursorControl Then
    _buffer(0).type = INPUT_MOUSE
    _buffer(0).mi.dx = CInt(Fix(warpedX * 65535.0f / _screenSize.Width))
    _buffer(0).mi.dy = CInt(Fix(warpedY * 65535.0f / _screenSize.Height))
    _buffer(0).mi.mouseData = 0
    _buffer(0).mi.dwFlags = MOUSEEVENTF_ABSOLUTE Or MOUSEEVENTF_MOVE
    _buffer(0).mi.time = 0
    _buffer(0).mi.dwExtraInfo = CType(0, IntPtr)

    _buffer(1).type = INPUT_MOUSE
    _buffer(1).mi.dx = 0
    _buffer(1).mi.dy = 0
    _buffer(1).mi.mouseData = 0
    _buffer(1).mi.dwFlags = MOUSEEVENTF_LEFTDOWN
    _buffer(1).mi.time = 10
    _buffer(1).mi.dwExtraInfo = CType(0, IntPtr)

    SendInput(2, _buffer, INPUT_SIZE)

  End If 'управление курсором

  Select Case _calibrationState
    Case 1
      _srcX(_calibrationState - 1) = x
      _srcY(_calibrationState - 1) = y
      _calibrationState = 2
      DoCalibration()
    Case 2
      _srcX(_calibrationState - 1) = x
      _srcY(_calibrationState - 1) = y
      _calibrationState = 3
      DoCalibration()
    Case 3
      _srcX(_calibrationState - 1) = x
      _srcY(_calibrationState - 1) = y
      _calibrationState = 4
      DoCalibration()
    Case 4
      _srcX(_calibrationState - 1) = x
      _srcY(_calibrationState - 1) = y
      _calibrationState = 5
      DoCalibration()
    Case Else
      Exit Select
  End Select 'калибровка 'кнопка нажата
Else
  If _cursorControl Then 'буксировка
     'инициализировать сумму
     Dim sumX As Single = warpedX, sumY As Single = warpedY
     _smoothingCount += 1 'количество хранимых значений

    If _smoothingCount > _smoothingPoints Then
       _smoothingCount = _smoothingPoints
```

```
        End If

        For i As Integer = 0 To _smoothingPoints - 2
          'сдвинуть хранимые точки
          _smoothingX(i) = _smoothingX(i + 1)
          _smoothingY(i) = _smoothingY(i + 1)

          sumX += _smoothingX(i) 'adding sum
          sumY += _smoothingY(i)
        Next i

        'добавить новую точку в последнюю позицию
        _smoothingX(_smoothingPoints - 1) = warpedX
        _smoothingY(_smoothingPoints - 1) = warpedY

        'вычислить средние
        Dim smoothWarpedX As Single = sumX / (_smoothingCount)
        Dim smoothWarpedY As Single = sumY / (_smoothingCount)

        'буксировка в точку с усредненными координатами
        _buffer(0).type = INPUT_MOUSE
        _buffer(0).mi.dx = CInt(Fix(smoothWarpedX * 65535.0f / _
                                _screenSize.Width))
        _buffer(0).mi.dy = CInt(Fix(smoothWarpedY * 65535.0f / _
                                _screenSize.Height))
        _buffer(0).mi.mouseData = 0
        _buffer(0).mi.dwFlags = MOUSEEVENTF_ABSOLUTE Or MOUSEEVENTF_MOVE
        _buffer(0).mi.time = 0
        _buffer(0).mi.dwExtraInfo = CType(0, IntPtr)
        SendInput(1, _buffer, INPUT_SIZE)
      End If
    End If 'ИК-точка видна
  Else
    If _lastWiiState.IRState.IRSensors(0).Found Then 'кнопка отпущена
      If _cursorControl Then
        _buffer(0).type = INPUT_MOUSE
        _buffer(0).mi.dx = 0
        _buffer(0).mi.dy = 0
        _buffer(0).mi.mouseData = 0
        _buffer(0).mi.dwFlags = MOUSEEVENTF_LEFTUP
        _buffer(0).mi.time = 0
        _buffer(0).mi.dwExtraInfo = CType(0, IntPtr)
        _buffer(1).type = INPUT_MOUSE
        _buffer(1).mi.dx = 0
        _buffer(1).mi.dy = 0
        _buffer(1).mi.mouseData = 0
        _buffer(1).mi.dwFlags = MOUSEEVENTF_MOVE
        _buffer(1).mi.time = 0
        _buffer(1).mi.dwExtraInfo = CType(0, IntPtr)
        SendInput(2, _buffer, INPUT_SIZE)

        'очистить сглаженные данные при отпускании кнопки
        ResetCursorSmoothing()
      End If
```

```
    End If 'ИК-точка пропала
  End If

  _lastWiiState.IRState.IRSensors(0).Found = ws.IRState.IRSensors(0).Found
```

Во второй половине обработчика события мы реагируем на нажатия кнопок на пульте Wiimote и обновляем пользовательский интерфейс. Кнопка «A» на пульте запускает процедуру калибровки, чтобы пользователю не нужно было возвращаться к компьютеру после позиционирования пульта. Как и раньше, для обнаружения нажатия на кнопку мы сравниваем текущее состояние с предыдущим. Стрелки джойстика отображаются на клавиши со стрелками. Обновляется уровень заряда батарейки и состояние видимости ИК-точки. Последнее полезно для определения правильности позиционирования пульта и наличия помех ИК-лучу. Мы включили также метод `ResetCursorSmoothing`, который вызывается для очистки данных сглаживания при отпускании кнопки (см. примеры 9.25 и 9.26).

Пример 9.25. Вторая половина обработчика события WiimoteChanged (версия на C#)

```
if (!_lastWiiState.ButtonState.A && ws.ButtonState.A)
   BeginInvoke((MethodInvoker)delegate() { btnCalibrate.PerformClick(); });

_lastWiiState.ButtonState.A = ws.ButtonState.A;

if (!_lastWiiState.ButtonState.Up && ws.ButtonState.Up)
    keybd_event(VK_UP, 0x45, 0, 0);
if (_lastWiiState.ButtonState.Up && !ws.ButtonState.Up)
    keybd_event(VK_UP, 0x45, KEYEVENTF_KEYUP, 0);

_lastWiiState.ButtonState.Up = ws.ButtonState.Up;
if (!_lastWiiState.ButtonState.Down && ws.ButtonState.Down)
    keybd_event(VK_DOWN, 0x45, 0, 0);
if (_lastWiiState.ButtonState.Down && !ws.ButtonState.Down)
    keybd_event(VK_DOWN, 0x45, KEYEVENTF_KEYUP, 0);

_lastWiiState.ButtonState.Down = ws.ButtonState.Down;
if (!_lastWiiState.ButtonState.Left && ws.ButtonState.Left)
    keybd_event(VK_LEFT, 0x45, 0, 0);
if (_lastWiiState.ButtonState.Left && !ws.ButtonState.Left)
    keybd_event(VK_LEFT, 0x45, KEYEVENTF_KEYUP, 0);

_lastWiiState.ButtonState.Left = ws.ButtonState.Left;
if (!_lastWiiState.ButtonState.Right && ws.ButtonState.Right)
    keybd_event(VK_RIGHT, 0x45, 0, 0);
if (_lastWiiState.ButtonState.Right && !ws.ButtonState.Right)
    keybd_event(VK_RIGHT, 0x45, KEYEVENTF_KEYUP, 0);
_lastWiiState.ButtonState.Right = ws.ButtonState.Right;

 BeginInvoke(new UpdateUIDelegate(UpdateUI), e.WiimoteState);
}

private void UpdateUI(WiimoteState ws)
```

```
{
  //нарисовать уровень заряда батарейки
  pbBattery.Value = (ws.Battery > 0xc8 ? 0xc8 : (int)ws.Battery);
  float f = (((100.0f * 48.0f * (float)(ws.Battery / 48.0f))) / 192.0f);
  lblBattery.Text = f.ToString("F");

  //отметить флажки, соответствующие видимым точкам
  String irstatus = "Visible IR dots: ";
  if (ws.IRState.IRSensors[0].Found)
     irstatus += "1 ";
  if (ws.IRState.IRSensors[1].Found)
     irstatus += "2 ";
  if (ws.IRState.IRSensors[2].Found)
     irstatus += "3 ";
  if (ws.IRState.IRSensors[3].Found)
     irstatus += "4 ";
  lblIRVisible.Text = irstatus;
}

private void ResetCursorSmoothing() //очистить данные сглаживания
{
  _smoothingCount = 0;
  for (int i = 0; i < 10; i++)
  {
    _smoothingX[i] = 0;
    _smoothingY[i] = 0;
  }
}
```

Пример 9.26. Вторая половина обработчика события WiimoteChanged (версия на VB)

```
  If (Not _lastWiiState.ButtonState.A) AndAlso ws.ButtonState.A Then
      BeginInvoke(CType(AddressOf CalibrateClick, MethodInvoker))
  End If

  _lastWiiState.ButtonState.A = ws.ButtonState.A

  If (Not _lastWiiState.ButtonState.Up) AndAlso ws.ButtonState.Up Then
      keybd_event(VK_UP, &H45, 0, 0)
  End If
  If _lastWiiState.ButtonState.Up AndAlso (Not ws.ButtonState.Up) Then
      keybd_event(VK_UP, &H45, KEYEVENTF_KEYUP, 0)
End If
_lastWiiState.ButtonState.Up = ws.ButtonState.Up

  If (Not _lastWiiState.ButtonState.Down) AndAlso ws.ButtonState.Down Then
      keybd_event(VK_DOWN, &H45, 0, 0)
  End If
  If _lastWiiState.ButtonState.Down AndAlso (Not ws.ButtonState.Down) Then
      keybd_event(VK_DOWN, &H45, KEYEVENTF_KEYUP, 0)
  End If
  _lastWiiState.ButtonState.Down = ws.ButtonState.Down
```

```
  If (Not _lastWiiState.ButtonState.Left) AndAlso ws.ButtonState.Left Then
      keybd_event(VK_LEFT, &H45, 0, 0)
  End If
  If _lastWiiState.ButtonState.Left AndAlso (Not ws.ButtonState.Left) Then
      keybd_event(VK_LEFT, &H45, KEYEVENTF_KEYUP, 0)
  End If
_lastWiiState.ButtonState.Left = ws.ButtonState.Left

  If (Not _lastWiiState.ButtonState.Right) AndAlso ws.ButtonState.Right Then
      keybd_event(VK_RIGHT, &H45, 0, 0)
  End If
  If _lastWiiState.ButtonState.Right AndAlso (Not ws.ButtonState.Right) Then
      keybd_event(VK_RIGHT, &H45, KEYEVENTF_KEYUP, 0)
  End If
  _lastWiiState.ButtonState.Right = ws.ButtonState.Right

  BeginInvoke(New UpdateUIDelegate(AddressOf UpdateUI), e.WiimoteState)
End Sub

Private Sub CalibrateClick()
  btnCalibrate.PerformClick()
End Sub

Private Sub UpdateUI(ByVal ws As WiimoteState)
  'нарисовать уровень заряда батарейки
  If ws.Battery > &Hc8 Then
    pbBattery.Value = (&Hc8)
  Else
    pbBattery.Value = (CInt(Fix(ws.Battery)))
  End If
  Dim f As Single = (((100.0f * 48.0f * CSng(ws.Battery / 48.0f))) / 192.0f)
  lblBattery.Text = f.ToString("F")

  'отметить флажки, соответствующие видимым точкам
  Dim irstatus As String = "Visible IR dots: "
  If ws.IRState.IRSensors(0).Found Then
    irstatus &= "1 "
  End If
  If ws.IRState.IRSensors(1).Found Then
    irstatus &= "2 "
  End If
  If ws.IRState.IRSensors(2).Found Then
    irstatus &= "3 "
  End If
  If ws.IRState.IRSensors(3).Found Then
    irstatus &= "4 "
  End If
  lblIRVisible.Text = irstatus
End Sub

Private Sub ResetCursorSmoothing() 'очистить данные сглаживания
  _smoothingCount = 0
  For i As Integer = 0 To 9
    _smoothingX(i) = 0
```

```
    _smoothingY(i) = 0
  Next i
End Sub
```

Нам нужно еще несколько вспомогательных методов для вычисления коэффициента калибровки и сохранения/загрузки данных калибровки в файле. Коэффициент калибровки (tracking utilization) вычисляется как отношение площади четырехугольника с вершинами в исходных точках к площади идеального четырехугольника, каковым является область изображения камеры (1024×768) за вычетом заданных полей с каждой стороны. Это тот четырехугольник, который получился бы, если бы пульт Wiimote находился прямо перед экраном, и все пиксели совместились бы.

В калибровочном файле хранятся только координаты четырех исходных точек, полученных от камеры, поскольку координаты конечных точек нам и так известны (это центры крестиков). При загрузке файла мы сначала проверяем, существует ли он, затем вычисляем матрицу преобразования и активируем управление курсором (примеры 9.27 и 9.28).

Пример 9.27. Вспомогательные методы (версия на C#)

```
void UpdateTrackingUtilization()
{
  //площадь идеально откалиброванной области (совпадающей с экраном)
  float idealArea = (1 - 2*_calibrationMargin) * 1024 *
                    (1 - 2*_calibrationMargin) * 768;

  //площадь четырехугольника
  float actualArea = 0.5f * Math.Abs((_srcX[1] - _srcX[2]) *
                                     (_srcY[0] - _srcY[3]) -
                                     (_srcX[0] - _srcX[3]) *
                                     (_srcY[1] - _srcY[2]));

  float util = (actualArea / idealArea)*100;
  string utilstatus = "Tracking Utilization: " + util.ToString("f0")+"%";

  BeginInvoke(new UpdateTrackingUtilizationDelegate(
              UpdateTrackingUtilizationUI), utilstatus);
}

private void UpdateTrackingUtilizationUI(string utilStatus)
{
  lblTrackingUtil.Text = utilStatus;
}

public void LoadCalibrationData()
{
  // создать объект-читатель и открыть файл
  try
  {
    TextReader tr = new StreamReader(CalibrationFilename);
```

```
    for (int i = 0; i < 4; i++)
    {
      _srcX[i] = float.Parse(tr.ReadLine());
      _srcY[i] = float.Parse(tr.ReadLine());
    }

    // закрыть поток
    tr.Close();
  }
  catch (FileNotFoundException)
  {
    //данных о калибровке нет
    return;
  }

  _warper.SetDestination( _screenSize.Width * _calibrationMargin,
                          _screenSize.Height * _calibrationMargin,
                          _screenSize.Width * (1.0f-_calibrationMargin),
                          _screenSize.Height * _calibrationMargin,
                          _screenSize.Width * _calibrationMargin,
                          _screenSize.Height * (1.0f - _calibrationMargin),
                          _screenSize.Width * (1.0f - _calibrationMargin),
                          _screenSize.Height * (1.0f - _calibrationMargin));
  _warper.SetSource(_srcX[0], _srcY[0], _srcX[1], _srcY[1], _srcX[2],
                    _srcY[2], _srcX[3], _srcY[3]);
  _warper.ComputeWarp();
  _cursorControl = true;
  cbCursorControl.Checked = _cursorControl;

  UpdateTrackingUtilization();
}

public void SaveCalibrationData()
{
  TextWriter tw = new StreamWriter(CalibrationFilename);

  // выводим в файл одну строку текста
  for (int i = 0; i < 4; i++)
  {
    tw.WriteLine(_srcX[i]);
    tw.WriteLine(_srcY[i]);
  }

  // закрываем поток
  tw.Close();
}
```

Пример 9.28. Вспомогательные методы (версия на VB)

```
Private Sub UpdateTrackingUtilization()
  'площадь идеально откалиброванной области (совпадающей с экраном)
  Dim idealArea As Single = (1 - 2*_calibrationMargin) * 1024 * _
                            (1 - 2*_calibrationMargin) * 768

  'площадь четырехугольника
```

```
  Dim actualArea As Single = 0.5f * Math.Abs((_srcX(1) - _srcX(2)) * _
                                             (_srcY(0) - _srcY(3)) - _
                                             (_srcX(0) - _srcX(3)) * _
                                             (_srcY(1) - _srcY(2)))

  Dim util As Single = (actualArea / idealArea)*100
  Dim utilstatus As String = "Tracking Utilization: " & util.ToString("f0") & "%"

  BeginInvoke(New UpdateTrackingUtilizationDelegate(AddressOf _
              UpdateTrackingUtilizationUI), utilstatus)
End Sub

Private Sub UpdateTrackingUtilizationUI(ByVal utilStatus As String)
  lblTrackingUtil.Text = utilStatus
End Sub

Public Sub LoadCalibrationData()
  ' создать объект-читатель и открыть файл
  Try
    Dim tr As TextReader = New StreamReader(CalibrationFilename)
    For i As Integer = 0 To 3
        _srcX(i) = Single.Parse(tr.ReadLine())
        _srcY(i) = Single.Parse(tr.ReadLine())
    Next i

    ' закрыть поток
    tr.Close()
  Catch e1 As FileNotFoundException
    'данных о калибровке нет
    Return
  End Try

  _warper.SetDestination(_screenSize.Width * _calibrationMargin, _
                         _screenSize.Height * _calibrationMargin, _
                         _screenSize.Width * (1.0f-_calibrationMargin), _
                         _screenSize.Height * _calibrationMargin, _
                         _screenSize.Width * _calibrationMargin, _
                         _screenSize.Height * (1.0f - _calibrationMargin), _
                         _screenSize.Width * (1.0f - _calibrationMargin), _
                         _screenSize.Height * (1.0f - _calibrationMargin))
  _warper.SetSource(_srcX(0), _srcY(0), _srcX(1), _srcY(1), _srcX(2),_
                    _srcY(2), _srcX(3), _srcY(3))
  _warper.ComputeWarp()
  _cursorControl = True
  cbCursorControl.Checked = _cursorControl

  UpdateTrackingUtilization()
End Sub

Public Sub SaveCalibrationData()
  Dim tw As TextWriter = New StreamWriter(CalibrationFilename)

  ' выводим в файл одну строку текста
  For i As Integer = 0 To 3
      tw.WriteLine(_srcX(i))
```

```
            tw.WriteLine(_srcY(i))
        Next i

        ' закрываем поток
        tw.Close()
    End Sub
```

Все наконец-то! Программа устанавливает соединение с пультом Wiimote, показывает окно калибровки, собирает калибровочные данные, вычисляет матрицу проективного преобразования пикселов камеры в пиксели экрана, сохраняет калибровочные данные в файле и эмулирует мышь. Теперь у вас есть собственная довольно эффективная и недорогая интерактивная электронная доска.

Использование программы

Чтобы воспользоваться написанной программой, найдите такое место для пульта Wiimote, чтобы камера видела ваш экран. Угол обзора камеры составляет 45 градусов. Имейте это в виду и постарайтесь расположить пульт как можно ближе к экрану, но так, чтобы он целиком попадал в поле зрения камеры. Индикаторы видимых точек (Visible IR dots) помогут вам понять, видит ли Wiimote весь экран. Активировав маркеры вблизи каждого угла экрана, убедитесь, что камера их видит. Если точки видны, диагностические элементы это покажут. Начните калибровку и включайте светодиод в центре каждого крестика. Программа должна автоматически переходить к следующему крестику. Когда все четыре крестика будут обработаны, окно калибровки исчезнет. Теперь ИК-перо должно управлять курсором мыши, и вы можете приступить к работе с интерактивной доской.

Качество слежения сильно зависит от положения пульта Wiimote. Если он находится слишком далеко от экрана или под слишком острым углом, то поле зрения камеры недостаточно, и качество слежения будет плохим. Вы можете воспользоваться диагностическим показателем «Tracking Utilization» после каждой калибровки, чтобы оценить удачность расположения пульта. Чем выше значение, тем лучше будет отслеживаться перемещение ИК-точки. Получить 100%-ный коэффициент и одновременно удобные условия для работы крайне сложно, поскольку это означает, что пульт находится прямо перед экраном. Но если коэффициент меньше 50%, то попробуйте найти для пульта другое место. Кроме того, нужно следить за тем, чтобы не перекрывать пульту обзор ИК-указки. Если ваша рука или какая-то другая часть тела окажутся перед камерой, то она не сможет увидеть точку, следовательно, отслеживание ее перемещения будет прервано. Конфигурация, показанная на рис. 9.6, дает неплохое качество слежения и с большой вероятностью не будет загораживать камеру. Если держать перо так, чтобы бы его кончик указывал в сторону от вас, то шансы случайно перекрыть обзор камере тоже уменьшаются.

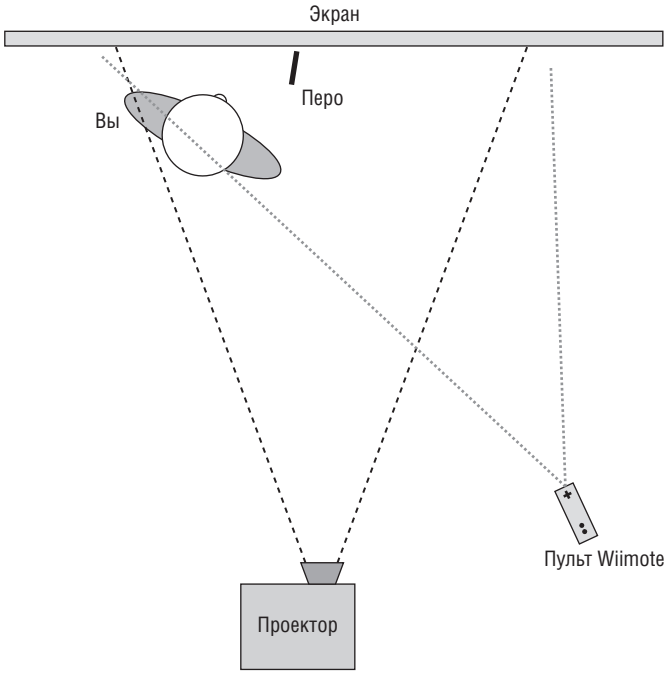

Рис. 9.6. *Типичное расположение электронной доски с фронтальной проекцией для программы WiimoteWhiteboard*

Если вы настолько усердны, что готовы сконструировать экран с обратной проекцией, то можете поместить Wiimote позади экрана рядом с проектором и получить очень высокое качество слежения, вообще не думая о блокировке обзора камеры. Выше уже отмечалось, что если проектора у вас нет, то можно направить пульт почти на любой плоский монитор, и слежение будет работать – вы получите что-то вроде планшетного ПК.

Примечание

Комбинация этого приложения с проектором фактически позволяет превратить чуть ли не любую плоскую поверхность в интерактивную доску. Это может быть стена, стол, пол, потолок, даже ваша рубашка (при условии, что вы не будете слишком активно перемещаться). Получилась забавная и достаточно гибкая программулька!

Заключительные замечания

Хотя в виде экспериментальных образцов системы типа multi-touch существовали с 1970-х годов, их коммерческий дебют состоялся лишь недавно. Термин «multi-touch» относится к интерактивным системам, способным одновременно распознавать несколько касаний. Поскольку у нас две руки и 10 пальцев, а иногда мы работаем совместно с другими людьми, то такие системы дают естественный способ взаимодействия с цифровыми материалами, очень похожий на то, как мы обращаемся с реальными предметами. Поскольку в приложении Wiimote Whiteboard для слежения используется камера, встроенная в пульт Wiimote, то вы можете одновременно отслеживать до четырех точек. Следовательно, теоретически можно было бы работать сразу с четырьмя ИК-указками. Все необходимое для добавления дополнительных точек в написанной нами программе уже имеется. Нужно просто читать в обработчике события `WiimoteChanged` значения координат остальных точек из переменной `WiimoteState`. Однако обработку информации о дополнительных точках вам придется взять на себя. Поскольку современные версии Microsoft Windows поддерживают лишь одну мышь, не существует очевидного соответствия между несколькими указками и одним курсором. Приложения, ориентированные на одновременные касания или работу с несколькими курсорами, нужно с самого начала проектировать с учетом нескольких источников ввода. В настоящее время такие приложения – редкость. Но, разумеется, никто не мешает вам создать собственное multitouch-приложение, основываясь на том, чему вы научились в этой главе. Удачи!

10

Анимированное музыкальное светошоу

Автор	Брайан Пик
Сложность	Средняя
Необходимое время	5 часов
Стоимость	$80 за интерфейсную плату Phidget Interface Kit 0/0/4 плюс стоимость удлинительных шнуров, проводов и гирлянды лампочек
Программное обеспечение	Visual C# 2008 Express, Visual Basic 2008 Express или любая полная редакция Visual Studio 2008
Оборудование	Интерфейсная плата Phidget Interface Kit 0/0/4 (одна плата может управлять четырьмя отдельными гирляндами); пять удлинительных шнуров с двухштырьковыми вилками для использования вне помещения; провод калибра 14–16; две муфты под провод калибра 14–16; корпус, достаточно большой для размещения одной или нескольких интерфейсных плат (продается в магазине радиотоваров); гирлянды лампочек; внешние динамики или FM-передатчик, чтобы можно было слушать музыку
Адрес в Интернете	*http://www.c4fbook.com/HolidayLights*

Кто не видел видеозапись с каким-нибудь домом, обвешанным новогодними лампочками, которые мигают в такт музыке? Пивоваренная компания Miller Brewing Company даже пускала по телевизору коммерческий ролик с таким светошоу в 2006 и 2007 году.

На первый взгляд может показаться, что организация подобного шоу – сложное дело, для которого нужно располагать обширными знаниями. Но на самом деле потребуется лишь интерфейсная плата Phidget Interface Kit, несколько удлинительных шнуров и компьютер.

В этом проекте мы покажем, как с помощью данных компонентов и доморощенной программы создать собственное внутреннее или наружное светошоу, синхронизированное с вашей любимой музыкой.

Краткий обзор

Архитектурно проект довольно прост. Во-первых, понадобится аппаратура, способная включать и выключать лампочки. Во-вторых, мы напишем приложение, которое будет повторять ритм проигрываемой песни для включения и выключения различных «каналов» (то есть гирлянд лампочек). После того как шоу будет «записано», мы сможем воспроизвести ритм синхронно с песней. Кроме того, мы добавим библиотеку MIDI, которая позволит импортировать музыкальный MIDI-файл и использовать хранящиеся в файле данные для управления светошоу.

Подготовка аппаратуры

Аппаратная часть строится на базе интерфейсной платы Phidget Interface Kit 0/0/4, содержащей четыре релейных выхода, которые могут переключаться при подаче постоянного или переменного тока. Приведенные в настоящей главе инструкции рассчитаны на сборку одного узла, управляющего четырьмя световыми каналами, но никто не мешает собрать и несколько узлов для организации более масштабного светошоу.

На основе интерфейсной платы мы соберем устройство, которое включается в одну розетку переменного тока и дает четыре выходных розетки, включаемых и выключаемых с помощью реле. Если в эти розетки воткнуть световые гирлянды, то их можно будет включать и выключать по нашему желанию. Описываемая в этой главе программа позволит создать «последовательность» включения и выключения гирлянд в заданные моменты времени, обычно синхронно с музыкой. Тем самым мы получим анимированное светошоу.

Первым делом подготовим удлинительные шнуры. Отрежьте от шнура конец с розеткой. Затем разделите жилы на отрезке длиной примерно 5 см. Очистите обе жилы от изоляции, затем пальцами аккуратно скрутите их, как показано на рис. 10.1. Удалять изоляцию лучше кусачками, которые можно купить в ближайшем магазине (если в хозяйстве нет). Нож или ножницы тоже подойдут, но ими работать гораздо труднее.

Рис. 10.1. *Подготовленный конец удлинителя, с которого срезана розетка*

Затем отрежьте вилки с четырех оставшихся шнуров. Снова разделите шнур на две жилы, очистите от изоляции и скрутите оголенные концы, как показано на рис. 10.2.

Теперь отрежьте четыре одинаковых куска провода калибра 14–16 (диаметром от 1.291 до 1.628 мм). Длина каждого куска должна составлять от 5 до 10 см. Очистите концы от изоляции и скрутите их, как и выше.

Рис. 10.2. *Подготовленный конец удлинителя, с которого срезана вилка*

Следующий шаг самый важный. Жилы удлинителей покрыты двумя разными видами резиновой изоляции. Один провод ребристый, другой – гладкий. Ребристый провод называется нейтральным (ноль), он должен подходить к «толстому» штырьку вилки; гладкий называется токопроводящим (фаза) и должен подходить к меньшему штырьку. Разница показана на рис. 10.3.

Рис. 10.3. *Гладкий и ребристый провода*

Имеет смысл как-нибудь промаркировать каждый провод, чтобы потом не перепутать. Очень важно выполнять следующие шаги в строгом соответствии с инструкцией, используя именно те провода, что указаны.

Возьмите четыре коротких отрезка провода и скрутите их вместе с ребристой жилой (ноль) того удлинительного шнура, на котором осталась вилка (см. рис. 10.4).

Наденьте на скрученные провода муфту, чтобы они не распались и были закрыты. Оголенные провода должны находиться внутри муфты, снаружи остаются только изолированные части (см. рис. 10.5).

Затем скрутите вместе гладкие жилы (фазы) четырех удлинителей, на которых остались розетки, с гладкой жилой (фазой) удлинителя, на котором осталась вилка (то есть с другой жилой удлинителя, использованного выше), как показано на рис. 10.6.

Рис. 10.4. *Четыре коротких провода, скрученные с нейтральной жилой удлинителя*

Рис. 10.5. *Скрученные провода упрятаны под муфту*

Рис. 10.6. *Четыре «фазы», скрученные с нейтральной жилой*

Рис. 10.7. *«Фазы», скрученные с нейтральной жилой и убранные в муфту*

Опять-таки наденьте муфту, убрав в нее оголенные провода, как показано на рис. 10.7.

Если вы все сделали правильно, то вне муфт должны остаться концы четырех отрезков провода и четыре ребристые (нейтральные) жилы удлинителей, заканчивающиеся розетками. Их мы прикрутим к винтовым клеммам платы Phidget Interface Kit 0/0/4.

На плате Phidget Interface Kit 0/0/4 есть четыре группы винтовых клемм. В каждой группе три клеммы, обозначенные NO, NC и *X*C, где *X* – связанное с клеммой реле. Эти аббревиатуры означают соответственно «Normally Open» (нормально разомкнутый), «Normally Closed» (нормально замкнутый) и «Common» (общая земля).

В этом проекте лампочки в нормальном положении должны быть выключены и включаются программно, поэтому мы будем использовать порты NO и XC. Не имеет значения, к какой клемме каждой группы какой провод подсоединен; важно лишь, чтобы в каждой группе был один короткий провод и один удлинитель, как показано на рис. 10.8.

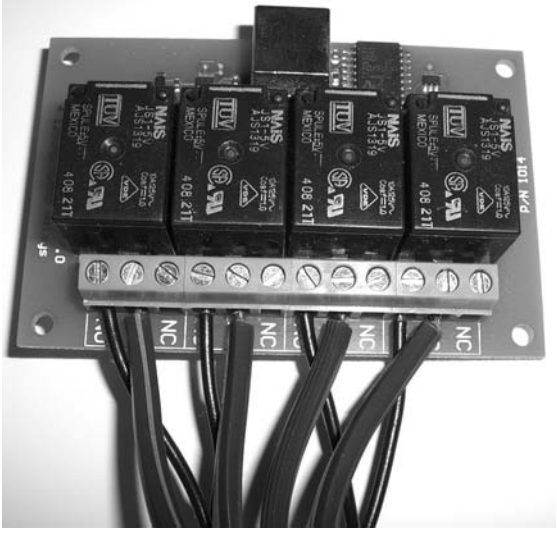

Рис. 10.8. *Подключение проводов к плате Phidget Interface Kit 0/0/4*

На рис. 10.9 показана простая электрическая схема получившегося подключения.

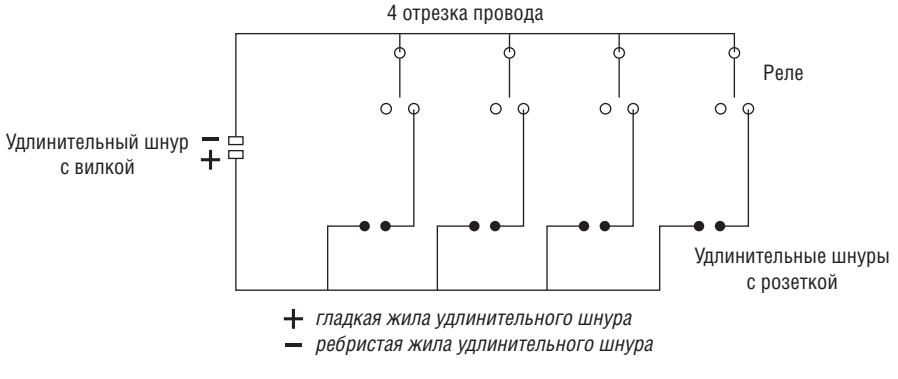

Рис. 10.9. *Электрическая схема одной интерфейсной платы с подключенными проводами*

Чтобы все выглядело аккуратно, интерфейсные платы следует поместить в какой-нибудь корпус. Я собрал две платы и поместил их в большую коробку. Просверлив отверстия в коротких сторонах коробки, я протащил через них USB-разъемы. Кроме того, я сделал на обеих длинных сторонах выемки для пучков проводов. Обе платы были надежно зафиксированы внутри коробки кусочками пенопласта. Если вы собираетесь устраивать шоу вне дома, то позаботьтесь о водонепроницаемости, чтобы ни дождь, ни снег не попали внутрь и не вывели электронику из строя! Готовое изделие показано на рис. 10.10 и 10.11.

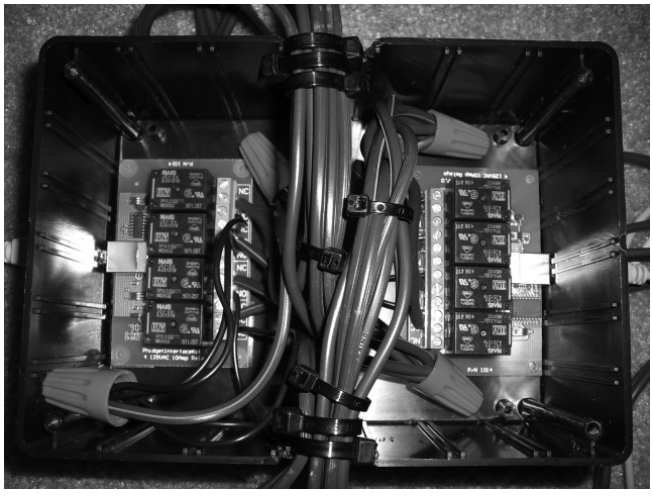

Рис. 10.10. *Расположение деталей внутри корпуса*

Рис. 10.11. *Собранное устройство с двумя интерфейсными платами, обеспечивающими восемь каналов управления*

О силе тока

Световые гирлянды будут включаться в розетки удлинителей. Средняя гирлянда, состоящая из миниатюрных лампочек, потребляет примерно 0.3 ампер. Гирлянда с лампочками побольше может потреблять 1–2 А. Реле на плате Phidget рассчитано на переменный ток 10 А при напряжении 250 В (или примерно на 20 А при напряжении 110 В).

Стандартная домашняя электросеть способна выдержать ток 15–20 А (потом выбьет предохранители). На автомате-прерывателе у вас дома должна быть указана максимально допустимая сила тока в розетках. Не забывайте, что другие устройства, подключенные к сети, тоже потребляют мощность, это следует учитывать, рассчитывая общую силу тока.

Подведем итоги: рассчитывайте нагрузку так, чтобы сила тока на каждом канале платы Phidget Interface Kit не превышала 10 А[1], а общая сила тока с учетом всех дополнительных потребителей в той же сети оставалась в пределах 15–20 А.

Программное обеспечение

Для управления интерфейсными платами мы создадим приложение, которое позволит пользователю сохранять последовательность нажатий клавиш, соответствующих моментам включения и выключения света, а затем воспроизводить ее без вмешательства человека.

Требования к ПО

Прежде всего, необходимо установить библиотеку Phidget API, с помощью которой вы будете обращаться к плате Phidget Interface Kit. Она часто обновляется, и иногда новые версии оказываются несовместимы с предыдущими. Поэтому на сайте *http://www.c4fbook.com/*, а также по адресу *http://www.oreilly.com/catalog/9780596520748* выложена версия, которой мы пользовались в этом проекте. Если вы еще не установили ее, сделайте это сейчас.

Реализация ПО

Выше мы уже отмечали, что прилагаемая к этому проекту программа позволяет отбивать ритм на клавиатуре, слушая песню. Тем самым вы отмечаете моменты, когда нужно включать и выключать отдельные гирлянды. Для этого нам нужен механизм запоминания нажатий клавиш и точного измерения временных интервалов. Кроме того, нужно как-то визуализировать эти данные, сохранять их для последующего воспроизведения и воспроизводить в точном соответствии с исходным

[1] 20 А при напряжении 110 В, применяемом в бытовых электросетях в США. – *Примеч. перев.*

ритмом. В следующих разделах мы поговорим обо всех этих составляющих приложения Light Sequencer. Само приложение намного объемнее представленного в этой главе. Здесь мы сможем обсудить лишь самые важные его части. Полный исходный текст можно найти на сопроводительном сайте книги по адресу *http://www.c4fbook.com/HolidayLights*.

Класс Sequence

Основной класс, с которым мы будем работать в этом приложении, называется `Sequence`. Он содержит путь к музыкальному файлу, который мы хотим проиграть (приложение поддерживает форматы WAV, MP3 и MIDI), кое-какую описательную информацию о песне (название, исполнитель), ее продолжительность, интервал между метками на главной шкале (об этом чуть позже) и список объектов `Channel`. Каждый объект `Channel` (канал) представляет один выходной порт на интерфейсной плате Phidget Interface Kit. Заготовка класса `Sequence` показана на рис. 10.1 и 10.2.

Пример 10.1. Переменные-члены и конструкторы класса Sequence (версия на C#)

```
public enum MusicType
{
  Sample = 0,
  MIDI = 1
}

public class Sequence
{
  private const string FILE_VERSION = "2";

  public string Title;
  public string Artist;
  public string MusicFile;
  public int MusicLength;
  public float Interval; // в мс
  public int Version = int.Parse(FILE_VERSION);
  public List<Channel> Channels = new List<Channel>();
  public MusicType MusicType;

  public Sequence() : this(null)
  {
  }

  public Sequence(string filename)
  {
    if(!string.IsNullOrEmpty(filename))
       Load(filename);
  }
}
```

Пример 10.2. Переменные-члены и конструкторы класса Sequence (версия на VB)

```
Public Enum MusicType
  Sample = 0
  MIDI = 1
End Enum

Public Class Sequence
  Private Const FILE_VERSION As String = "2"

  Public Title As String
  Public Artist As String
  Public MusicFile As String
  Public MusicLength As Integer
  Public Interval As Single ' в мс
  Public Version As Integer = Integer.Parse(FILE_VERSION)
  Public Channels As List(Of Channel) = New List(Of Channel)()
  Public MusicType As MusicType

  Public Sub New()
    Me.New(Nothing)
  End Sub

  Public Sub New(ByVal filename As String)
    If (Not String.IsNullOrEmpty(filename)) Then
      Load(filename)
    End If
  End Sub
End Sequence
```

Объект `Channel` содержит серийный номер конкретной платы Phidget Interface Kit (чтобы ее можно было адресовать напрямую); номер выходного порта, на который отображен канал; целое число, представляющее канал в последовательности; целое число, соответствующее конкретной дорожке в MIDI-файле (об этом позже), и собственно ритмические данные для этого канала. Код класса `Channel` приведен в примерах 10.3 и 10.4.

Пример 10.3. Класс Channel (версия на C#)

```
public class Channel
{
  // серийный номер платы phidget, на которую отображен этот канал
  private int _serialNumber;

  // индекс в массиве выходных портов платы phidget,
  // соответствующий этому каналу
  private int _outputIndex;

  // сквозной номер канала
  private int _channelNumber;

  private int _MIDIChannel;
```

```
// массив состояний ВЫКЛ/ВКЛ для каждого тика
private bool[] _data;

public Channel(int channelNumber)
{
  _channelNumber = channelNumber;
}

public Channel(int channelNumber, int serialNumber, int outputIndex)
{
  _channelNumber = channelNumber;
  _serialNumber = serialNumber;
  _outputIndex = outputIndex;
}

public Channel(int channelNumber, int serialNumber, int outputIndex,
               int midiChannel, int dataLength)
{
  _channelNumber = channelNumber;
  _serialNumber = serialNumber;
  _outputIndex = outputIndex;
  _MIDIChannel = midiChannel;
  _data = new bool[dataLength];
}

public Channel(int channelNumber, int serialNumber, int outputIndex,
               int dataLength)
{
  _channelNumber = channelNumber;
  _serialNumber = serialNumber;
  _outputIndex = outputIndex;
  _data = new bool[dataLength];
}

public int Number
{
  get { return _channelNumber; }
  set { _channelNumber = value; }
}

public int SerialNumber
{
  get { return _serialNumber; }
  set { _serialNumber = value; }
}

public int OutputIndex
{
  get { return _outputIndex; }
  set { _outputIndex = value; }
}

public int MIDIChannel
{
```

```
    get { return this._MIDIChannel; }
    set { this._MIDIChannel = value; }
  }

  public bool[] Data
  {
    get { return _data; }
    set { _data = value; }
  }
}
```

Пример 10.4. Класс Channel (версия на VB)

```
Public Class Channel
  ' серийный номер платы phidget, на которую отображен этот канал
  Private _serialNumber As Integer

  ' индекс в массиве выходных портов платы phidget,
  ' соответствующий этому каналу
  Private _outputIndex As Integer

  ' сквозной номер канала
  Private _channelNumber As Integer

  Private _MIDIChannel As Integer

  ' массив состояний ВЫКЛ/ВКЛ для каждого тика
  Private _data() As Boolean

  Public Sub New(ByVal channelNumber As Integer)
    _channelNumber = channelNumber
  End Sub

  Public Sub New(ByVal channelNumber As Integer, _
                 ByVal serialNumber As Integer, _
                 ByVal outputIndex As Integer)
    _channelNumber = channelNumber
    _serialNumber = serialNumber
    _outputIndex = outputIndex
  End Sub

  Public Sub New(ByVal channelNumber As Integer, _
                 ByVal serialNumber As Integer, _
                 ByVal outputIndex As Integer, _
                 ByVal midiChannel As Integer, _
                 ByVal dataLength As Integer)
    _channelNumber = channelNumber
    _serialNumber = serialNumber
    _outputIndex = outputIndex
    _MIDIChannel = midiChannel
    _data = New Boolean(dataLength - 1){}
  End Sub

  Public Sub New(ByVal channelNumber As Integer, _
                 ByVal serialNumber As Integer, _
                 ByVal outputIndex As Integer, _
```

```
                ByVal dataLength As Integer)
    _channelNumber = channelNumber
    _serialNumber = serialNumber
    _outputIndex = outputIndex
    _data = New Boolean(dataLength - 1){}
  End Sub

  Public Property Number() As Integer
    Get
      Return _channelNumber
    End Get
    Set(ByVal value As Integer)
      _channelNumber = value
    End Set
  End Property

  Public Property SerialNumber() As Integer
    Get
      Return _serialNumber
    End Get
    Set(ByVal value As Integer)
        _serialNumber = value
    End Set
  End Property

  Public Property OutputIndex() As Integer
    Get
      Return _outputIndex
    End Get
    Set(ByVal value As Integer)
        _outputIndex = value
    End Set
  End Property

  Public Property MIDIChannel() As Integer
    Get
      Return Me._MIDIChannel
    End Get
    Set(ByVal value As Integer)
        Me._MIDIChannel = value
    End Set
  End Property

  Public Property Data() As Boolean()
    Get
      Return _data
    End Get
    Set(ByVal value As Boolean())
        _data = value
    End Set
  End Property
End Class
```

С помощью этих двух объектов мы можем представить всю последовательность и данные, необходимые для ее сохранения, загрузки и воспроизведения.

Сохраненная последовательность представляется в виде XML-файла. Код сохранения приведен в примерах 10.5 и 10.6.

Пример 10.5. Сохранение объекта Sequence (версия на C#)

```
public void Save(string filename)
{
  // выводить с отступами
  XmlWriterSettings settings = new XmlWriterSettings();
  settings.Indent = true;

  // создать новый XML-файл
  XmlWriter xml = XmlWriter.Create(filename, settings);

  // вывести начальные теги
  xml.WriteStartDocument();

  SaveSequence(xml);

  // вывести конечные теги
  xml.WriteEndDocument();

  // все закрываем
  xml.Flush();
  xml.Close();
}

private void SaveSequence(XmlWriter xml)
{
  StringBuilder sb = new StringBuilder();

  xml.WriteComment("Sequence file for LightSequencer");

  xml.WriteStartElement("sequence");

    // выводим основную конфигурационную информацию
    xml.WriteStartElement("config");
        xml.WriteElementString("fileVersion", FILE_VERSION);
        xml.WriteElementString("musicFile", this.MusicFile);
        xml.WriteElementString("musicLength", this.MusicLength.ToString());
        xml.WriteElementString("numChannels", this.Channels.Count.ToString());
        xml.WriteElementString("interval", this.Interval.ToString());
    xml.WriteEndElement();

    // выводим информацию о песне
    xml.WriteStartElement("musicInfo");
        xml.WriteElementString("title", this.Title);
        xml.WriteElementString("artist", this.Artist);
    xml.WriteEndElement();

    // выводим каналы
    xml.WriteStartElement("channels");
    foreach(Channel channel in this.Channels)
```

```
        {
          // 1==вкл, 0==выкл...корявенько, но работает. :)
          foreach(bool b in channel.Data)
            sb.Append((b ? "1" : "0"));

          // выводим информацию о канале
          xml.WriteStartElement("channel");
              xml.WriteAttributeString("number", channel.Number.ToString());
              xml.WriteAttributeString("serialNumber",
                                       channel.SerialNumber.ToString());
              xml.WriteAttributeString("outputIndex",
                                        channel.OutputIndex.ToString());
              xml.WriteAttributeString("midiChannel",
                                       channel.MIDIChannel.ToString());
              xml.WriteString(sb.ToString());
          xml.WriteEndElement();
          sb.Remove(0, sb.Length);
        }
        xml.WriteEndElement();
    xml.WriteEndElement();
}
```

Пример 10.6. Сохранение объекта Sequence (версия на VB)

```
Public Sub Save(ByVal filename As String)
  ' выводить с отступами
  Dim settings As New XmlWriterSettings()
  settings.Indent = True

  ' создать новый XML-файл
  Dim xml As XmlWriter = XmlWriter.Create(filename, settings)

  ' вывести начальные теги
  xml.WriteStartDocument()
  SaveSequence(xml)

  ' вывести конечные теги
  xml.WriteEndDocument()

  ' все закрываем
  xml.Flush()
  xml.Close()
End Sub

Private Sub SaveSequence(ByVal xml As XmlWriter)
  Dim sb As New StringBuilder()

  xml.WriteComment("Sequence file for LightSequencer")

  xml.WriteStartElement("sequence")

    ' выводим основную конфигурационную информацию
    xml.WriteStartElement("config")
       xml.WriteElementString("fileVersion", FILE_VERSION)
       xml.WriteElementString("musicFile", Me.MusicFile)
       xml.WriteElementString("musicLength", Me.MusicLength.ToString())
```

```
        xml.WriteElementString("numChannels", Me.Channels.Count.ToString())
        xml.WriteElementString("interval", Me.Interval.ToString())
    xml.WriteEndElement()

    ' выводим информацию о песне
    xml.WriteStartElement("musicInfo")
        xml.WriteElementString("title", Me.Title)
        xml.WriteElementString("artist", Me.Artist)
    xml.WriteEndElement()

    ' выводим каналы
    xml.WriteStartElement("channels")

    For Each channel As Channel In Me.Channels
        ' 1==вкл, 0==выкл...корявенько, но работает. :)
        For Each b As Boolean In channel.Data
          If b Then
            sb.Append(("1"))
          Else
            sb.Append(("0"))
          End If
        Next b

        ' выводим информацию о канале
        xml.WriteStartElement("channel")
            xml.WriteAttributeString("number", channel.Number.ToString())
            xml.WriteAttributeString("serialNumber", _
                                     channel.SerialNumber.ToString())
            xml.WriteAttributeString("outputIndex", _
                                     channel.OutputIndex.ToString())
            xml.WriteAttributeString("midiChannel", _
                                     channel.MIDIChannel.ToString())
            xml.WriteString(sb.ToString())
        xml.WriteEndElement()

        sb.Remove(0, sb.Length)
      Next channel
      xml.WriteEndElement()
    xml.WriteEndElement()
End Sub
```

При загрузке XML-данные разбираются, и последовательность воссоздается в памяти, готовая к использованию (примеры 10.7 и 10.8).

Пример 10.7. Загрузка объекта Sequence (версия на C#)

```
public void Load(string filename)
{
  // загрузить XML-файл
  XmlDocument xmlDoc = new XmlDocument();
  xmlDoc.Load(filename);

  // разобрать основную конфигурационную информацию
  this.Version =
int.Parse(xmlDoc.SelectSingleNode("sequence/config/fileVersion").InnerText);
```

```
  switch(this.Version)
  {
    case 1:
      LoadV1Sequence(xmlDoc);
      break;
    default:
      LoadSequence(xmlDoc);
      break;
  }
}

public void LoadSequence(XmlDocument xmlDoc)
{
  XmlNode node;

  // разобрать основную конфигурационную информацию
  this.MusicFile = xmlDoc.SelectSingleNode("sequence/config/
                                          musicFile").InnerText;

  if(this.MusicFile.EndsWith(".mid"))
   this.MusicType = MusicType.MIDI;

  this.MusicLength =
int.Parse(xmlDoc.SelectSingleNode("sequence/config/musicLength").InnerText);
  this.Interval =
float.Parse(xmlDoc.SelectSingleNode("sequence/config/interval").InnerText);
  int numChannels =
int.Parse(xmlDoc.SelectSingleNode("sequence/config/numChannels").InnerText);

  this.Title = xmlDoc.SelectSingleNode("sequence/musicInfo/
                                       title").InnerText;
  this.Artist = xmlDoc.SelectSingleNode("sequence/musicInfo/
                                       artist").InnerText;

  this.Channels.Clear();

  // поместить каналы во внутренний список,
  // чтобы с ними было проще работать
  for(int i = 0; i < numChannels; i++)
  {
    node = xmlDoc.SelectSingleNode("sequence/channels/channel[@number='" +
                                   i.ToString() + "']");
    if(node != null)
    {
      int serial = int.Parse(node.Attributes["serialNumber"].Value);
      char[] data = node.InnerText.ToCharArray();

      // добавить канал во внутренний список
      this.Channels.Add(new Channel(i, serial,
                        int.Parse(node.Attributes["outputIndex"].Value),
                        int.Parse(node.Attributes["midiChannel"].Value),
                        data.Length));

      // прочитать данные
      for(int j = 0; j < data.Length; j++)
```

```
        this.Channels[i].Data[j] = (data[j] == '1');
    }
    else
      this.Channels.Add(new Channel(i));
  }
}
```

Пример 10.8. Загрузка объекта Sequence (версия на VB)

```
Public Sub Load(ByVal filename As String)
  ' загрузить XML-файл
  Dim xmlDoc As New XmlDocument()
  xmlDoc.Load(filename)

  ' разобрать основную конфигурационную информацию
  Me.Version = Integer.Parse( _
          xmlDoc.SelectSingleNode("sequence/config/fileVersion").InnerText)
  Select Case Me.Version
    Case 1
      LoadV1Sequence(xmlDoc)
    Case Else
      LoadSequence(xmlDoc)
  End Select
End Sub

Public Sub LoadSequence(ByVal xmlDoc As XmlDocument)
  Dim node As XmlNode
  ' разобрать основную конфигурационную информацию
  Me.MusicFile = xmlDoc.SelectSingleNode( _
                                    "sequence/config/musicFile").InnerText

  If Me.MusicFile.EndsWith(".mid") Then
      Me.MusicType = MusicType.MIDI
  End If

  Me.MusicLength = Integer.Parse( _
      xmlDoc.SelectSingleNode("sequence/config/musicLength").InnerText)
  Me.Interval = Single.Parse( _
      xmlDoc.SelectSingleNode("sequence/config/interval").InnerText)
  Dim numChannels As Integer = Integer.Parse( _
      xmlDoc.SelectSingleNode("sequence/config/numChannels").InnerText)

  Me.Title = xmlDoc.SelectSingleNode("sequence/musicInfo/title").InnerText
  Me.Artist = xmlDoc.SelectSingleNode("sequence/musicInfo/artist").InnerText
  Me.Channels.Clear()

  ' поместить каналы во внутренний список,
  ' чтобы с ними было проще работать
  For i As Integer = 0 To numChannels - 1
      node = xmlDoc.SelectSingleNode( _
          "sequence/channels/channel[@number='" & i.ToString() & "']")
      If node IsNot Nothing Then
          Dim serial As Integer = Integer.Parse( _
              node.Attributes("serialNumber").Value)
          Dim data() As Char = node.InnerText.ToCharArray()
```

```
            ' добавить канал во внутренний список
            Me.Channels.Add(New Channel(i, serial, _
                        Integer.Parse(node.Attributes("outputIndex").Value), _
                        Integer.Parse(node.Attributes("midiChannel").Value), _
                        data.Length))

            ' прочитать данные
            For j As Integer = 0 To data.Length - 1
                Me.Channels(i).Data(j) = (data(j) = "1"c)
            Next j
        Else
            Me.Channels.Add(New Channel(i))
        End If
    Next i
End Sub
```

Отображение сетки

Научившись сохранять и загружать последовательность, мы должны придумать, как визуализировать записанные данные на экране. Проще всего воспользоваться элементом `GridView`, в котором каждый канал Phidget занимает одну строку. При записи ритмической последовательности мы можем раскрашивать ячейки в те моменты, когда нажимается клавиша. Цвет будет индикатором включения или выключения канала. На рис. 10.12 показано, как это выглядит на экране.

Присмотревшись внимательнее, вы заметите, что сетка содержит два заголовка: в первом откладываются секунды, а во втором – метки, помечающие отдельные ячейки, соответствующие тикам с интервалом 50 миллисекунд (в одной секунде как раз 20 таких ячеек).

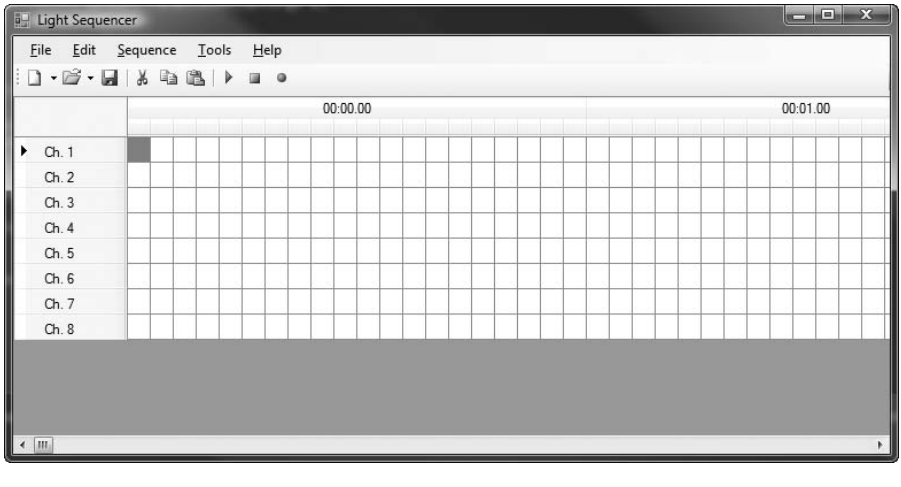

***Рис. 10.12.** Основной интерфейс программы Light Sequencer*

Слева по вертикали расположены надписи, обозначающие каждый канал интерфейсной платы (канал соответствует световой гирлянде). При горизонтальной прокрутке этот столбец остается неподвижным, а заголовки, отмечающие время, прокручиваются вместе с сеткой.

Для создания такого представления перетащите на поверхность формы два элемента DataGridView: один будет отображать только заголовок с отсчетом времени, другой – всю остальную сетку. Элемент DataGridView не поддерживает несколько заголовков, поэтому приходится идти на такую уловку.

Перетащив обе сетки на форму, мы можем создать временную шкалу с помощью кода из примеров 10.9 и 10.10.

Пример 10.9. Рисование сетки, представляющей последовательность (версия на C#)

```
private void CreateSequence()
{
  // привести сетку в исходное состояние
  dgvHeader.Rows.Clear();
  dgvHeader.Columns.Clear();
  dgvMain.Rows.Clear();
  dgvMain.Columns.Clear();

  // для ускорения сделать сетку невидимой на время рисования
  dgvMain.Visible = false;
  dgvHeader.Visible = false;

  // столбцов в 1 секунде = 1 секунда / временная протяженность
  // одного столбца
  _colsPerSec = 1000 / (int)_sequence.Interval;

  // нарисовать сетку
  // цикл по времени звучания
  for(int sec = 0; sec < _sequence.MusicLength; sec++)
  {
    // для каждой секунды из указанного промежутка
    // создать столбец в заголовке
    string header = "{0:0#}:{1:0#}.00";
    dgvHeader.Columns.Add(sec.ToString(), String.Format(header,
                          sec/60, sec%60));

    // это значение должно быть < 65535 для всей сетки (?)
    dgvHeader.Columns[sec].FillWeight = 0.0001f;

    // задать подходящую ширину столбца
    dgvHeader.Columns[sec].Width = _colsPerSec*COL_WIDTH;

    // теперь рисуем ячейки внутри одной секунды
    for(int partSec = 0; partSec < _colsPerSec; partSec++)
    {
      // без имени, задается FillWeight и устанавливается ширина
      dgvMain.Columns.Add("", "");
      dgvMain.Columns[(sec * _colsPerSec) + partSec].FillWeight = 0.0001f;
```

```
        dgvMain.Columns[(sec * _colsPerSec) + partSec].Width = COL_WIDTH;
      }
    }

    // добавляем заголовки строк (Ch. 1, Ch. 2 и т. д.)
    for(int channel = 0; channel < _sequence.Channels.Count; channel++)
    {
      // создать строку и текст заголовка
      DataGridViewRow dvr = new DataGridViewRow();
      dvr.HeaderCell.Value = "Ch. " + (channel+1);

      // запомнить сам объект Channel, он потом понадобится
      dvr.Tag = _sequence.Channels[channel];

      // добавить строку
      dgvMain.Rows.Add(dvr);
    }

    // установить ширину заголовков строк
    dgvMain.RowHeadersWidth = 100;
    dgvHeader.RowHeadersWidth = 100;

    // сделать сетку снова видимой
    dgvMain.Visible = true;
    dgvHeader.Visible = true;
  }
```

Пример 10.10. Рисование сетки, представляющей последовательность (версия на VB)

```
  Private Sub CreateSequence()
    ' привести сетку в исходное состояние
    dgvHeader.Rows.Clear()
    dgvHeader.Columns.Clear()
    dgvMain.Rows.Clear()
    dgvMain.Columns.Clear()

    ' для ускорения сделать сетку невидимой на время рисования
    dgvMain.Visible = False
    dgvHeader.Visible = False

    ' столбцов в 1 секунде = 1 секунда / временная протяженность одного столбца
    _colsPerSec = 1000 / CInt(Fix(_sequence.Interval))

    ' нарисовать сетку
    ' цикл по времени звучания
    For sec As Integer = 0 To _sequence.MusicLength - 1
      ' для каждой секунды из указанного промежутка создать столбец в заголовке
      Dim header As String = "{0:0#}:{1:0#}.00"
      dgvHeader.Columns.Add(sec.ToString(), _
                            String.Format(header, sec\60, sec Mod 60))

      ' это значение должно быть < 65535 для всей сетки (?)
      dgvHeader.Columns(sec).FillWeight = 0.0001f

      ' задать подходящую ширину столбца
```

```
    dgvHeader.Columns(sec).Width = _colsPerSec*COL_WIDTH

    ' теперь рисуем ячейки внутри одной секунды
    For partSec As Integer = 0 To _colsPerSec - 1
      ' без имени, задается FillWeight и устанавливается ширина
      dgvMain.Columns.Add("", "")
      dgvMain.Columns((sec * _colsPerSec) + partSec).FillWeight = 0.0001f
      dgvMain.Columns((sec * _colsPerSec) + partSec).Width = COL_WIDTH
    Next partSec
  Next sec

  ' добавляем заголовки строк (Ch. 1, Ch. 2 и т. д.)
  For channel As Integer = 0 To _sequence.Channels.Count - 1
    ' создать строку и текст заголовка
    Dim dvr As New DataGridViewRow()
    dvr.HeaderCell.Value = "Ch. " & (channel+1)

    ' запомнить сам объект Channel, он потом понадобится
    dvr.Tag = _sequence.Channels(channel)

    ' добавить строку
    dgvMain.Rows.Add(dvr)
  Next channel

  ' установить ширину заголовков строк
  dgvMain.RowHeadersWidth = 100
  dgvHeader.RowHeadersWidth = 100

  ' сделать сетку снова видимой
  dgvMain.Visible = True
  dgvHeader.Visible = True
End Sub
```

Нарисовав саму сетку, мы должны раскрасить ячейки в соответствии с записанной последовательностью. Для этого просто перебираем каналы и промежутки времени в каждом канале, закрашивая в синий цвет ячейки, соответствующие тем моментам, когда канал открыт (лампочки горят). См. примеры 10.11 и 10.12.

Пример 10.11. Представление последовательности в сетке (версия на C#)

```
private void RedrawSequence()
{
  for(int i = 0; i < _sequence.Channels.Count; i++)
  {
    for(int j = 0; j < _sequence.Channels[i].Data.Length; j++)
        dgvMain.Rows[i].Cells[j].Style.BackColor =
                (_sequence.Channels[i].Data[j]) ? Color.Blue : Color.White;
  }
}
```

Пример 10.12. Представление последовательности в сетке (версия на VB)

```
Private Sub RedrawSequence()
  For i As Integer = 0 To _sequence.Channels.Count - 1
```

```
    For j As Integer = 0 To _sequence.Channels(i).Data.Length - 1
      If (_sequence.Channels(i).Data(j)) Then
          dgvMain.Rows(i).Cells(j).Style.BackColor = Color.Blue
      Else
          dgvMain.Rows(i).Cells(j).Style.BackColor = Color.White
      End If
    Next j
    Next i
End Sub
```

Воспроизведение музыки

Научившись загружать, сохранять и отображать последовательность, мы должны теперь воспроизвести соответствующую ей музыку. Программа Light Sequencer поддерживает форматы WAV и MP3 (то есть оцифрованную музыку), а также MIDI-файлы.

Поскольку мы собираемся поддержать оба вида форматов, то создадим интерфейс, в котором опишем необходимые приложению методы, и реализуем этот интерфейс в одном классе для работы с WAV/MP3 и в другом классе для работы с MIDI. При такой организации мы сможем относительно легко добавить новый способ воспроизведения, если возникнет необходимость.

Определение интерфейса `IPlayback` приведено в примере 10.13 и 10.14.

Пример 10.13. Интерфейс IPlayback (версия на C#)

```
public interface IPlayback
{
  void Load(Sequence seq);
  void Unload();
  void Start();
  void Stop();
}
```

MIDI

MIDI расшифровывается как «Musical Instrument Digital Interface» (цифровой интерфейс музыкальных инструментов). В отличие от форматов MP3 или WAV, где сама песня закодирована в потоке данных, MIDI-файл содержит инструкции различным аппаратным компонентам о том, как воспроизвести музыку. Впоследствии этими данными можно воспользоваться для автоматического создания последовательности, синхронизированной с музыкой, так как нам точно известно, в какой момент какую ноту должен издать каждый инструмент.

Пример 10.14. Интерфейс IPlayback (версия на VB)

```
Public Interface IPlayback
  Sub Load(ByVal seq As Sequence)
  Sub Unload()
  Sub Start()
  Sub [Stop]()
End Interface
```

Воспроизвести WAV/MP3-файлы позволяет входящий в состав Win32 API интерфейс управления мультимедиа – MCI (Media Control Interface). MCI API дает возможность посылать текстовые команды, управляющие мультимедийными устройствами. В табл. 10.1 перечислены команды, которые нам понадобятся.

Таблица 10.1. Команды MCI для загрузки, выгрузки, воспроизведения и остановки

Команда	Строка
Загрузить мультимедийный файл	`open "filename.mp3" type mpegvideo alias MediaFile`
Выгрузить мультимедийный файл	`close MediaFile`
Воспроизвести файл	`play MediaFile from 0`
Остановить воспроизведение	`stop MediaFile`

Эти команды можно посылать мультимедийной аппаратуре с помощью функции `mciSendString`. Полный текст класса `MCIPlayback` для воспроизведения WAV/MP3-файлов приведен в примерах 10.15 и 10.16.

Пример 10.15. Класс MCIPlayback (версия на C#)

```
public class MCIPlayback : IPlayback
{
  // импортируем dll для воспроизведения музыки через MCI
  [DllImport("winmm.dll")]
  static extern Int32 mciSendString(String command, StringBuilder buffer,
                                    Int32 bufferSize, IntPtr hwndCallback);

  public void Load(Sequence seq)
  {
    string cmd = "open \"" + seq.MusicFile + "\" type mpegvideo alias
MediaFile";
    mciSendString(cmd, null, 0, IntPtr.Zero);
  }

  public void Unload()
  {
    string cmd = "close MediaFile";
    mciSendString(cmd, null, 0, IntPtr.Zero);
  }
```

```
  public void Start()
  {
    string cmd = "play MediaFile from 0";
    mciSendString(cmd, null, 0, IntPtr.Zero);
  }

  public void Stop()
  {
    string cmd = "stop MediaFile";
    mciSendString(cmd, null, 0, IntPtr.Zero);
  }
}
```

Пример 10.16. Класс MCIPlayback (версия на VB)

```
Public Class MCIPlayback
    Implements IPlayback
  ' импортируем dll для воспроизведения музыки через MCI
  <DllImport("winmm.dll")> _
  Shared Function mciSendString(ByVal command As String, _
                                ByVal buffer As StringBuilder,_
                                ByVal bufferSize As Int32, _
                                ByVal hwndCallback As IntPtr) As Int32
  End Function

  Public Sub Load(ByVal seq As Sequence) Implements IPlayback.Load
    Dim cmd As String = "open """ & seq.MusicFile & _
                        """ type mpegvideo alias MediaFile"
    mciSendString(cmd, Nothing, 0, IntPtr.Zero)
  End Sub

  Public Sub Unload() Implements IPlayback.Unload
    Dim cmd As String = "close MediaFile"
    mciSendString(cmd, Nothing, 0, IntPtr.Zero)
  End Sub

  Public Sub Start() Implements IPlayback.Start
    Dim cmd As String = "play MediaFile from 0"
    mciSendString(cmd, Nothing, 0, IntPtr.Zero)
  End Sub

  Public Sub [Stop]() Implements IPlayback.Stop
    Dim cmd As String = "stop MediaFile"
    mciSendString(cmd, Nothing, 0, IntPtr.Zero)
  End Sub
End Class
```

Для воспроизведения MIDI-файлов мы воспользуемся библиотекой MIDI Toolkit, которую написал Лесли Сэнфорд (Leslie Sanford) (*http://www.lesliesanford.com/*). Она позволяет очень легко загрузить MIDI-файл и воспроизвести его содержимое. Реализация интерфейса `IPlayback` в классе `MIDIPlayback` приведена в примерах 10.17 и 10.18.

Пример 10.17. Класс MIDIPlayback (версия на C#)

```
public class MIDIPlayback : IPlayback
{
  private Midi.Sequencer _MIDISequencer;
  private Midi.Sequence _MIDISequence;
  private Midi.OutputDevice _MIDIOutDevice;

  public MIDIPlayback()
  {
    _MIDISequencer = new Midi.Sequencer();
    _MIDISequence = new Midi.Sequence();

    // обработчики различных событий
    _MIDISequencer.ChannelMessagePlayed += sequencer_ChannelMessagePlayed;
    _MIDISequencer.Chased += sequencer_Chased;
    _MIDISequencer.Stopped += sequencer_Stopped;
  }

  public void Start()
  {
    _MIDISequencer.Start();
  }

  public void Load(Sequence seq)
  {
    // найти первое MIDI-устройство
    if(_MIDIOutDevice == null)
       _MIDIOutDevice = new Midi.OutputDevice(0);

    // загрузить MIDI-файл
    _MIDISequence.Load(seq.MusicFile);
    _MIDISequencer.Sequence = _MIDISequence;
  }

  public void Unload()
  {
    if(_MIDISequence != null)
       _MIDISequence.Clear();
    if(_MIDIOutDevice != null)
       _MIDIOutDevice.Close();
  }

  public void Stop()
  {
    _MIDISequencer.Stop();
  }

  void sequencer_Stopped(object sender, StoppedEventArgs e)
  {
    // послать сообщение "stop" звуковой карте
    foreach(ChannelMessage message in e.Messages)
    {
      if(!_MIDIOutDevice.IsDisposed)
          _MIDIOutDevice.Send(message);
```

```
    }
  }

  void sequencer_Chased(object sender,
                        Sanford.Multimedia.Midi.ChasedEventArgs e)
  {
    // послать сообщение "chased" звуковой карте
    foreach(ChannelMessage message in e.Messages)
    {
      if(!_MIDIOutDevice.IsDisposed)
          _MIDIOutDevice.Send(message);
    }
  }

  void sequencer_ChannelMessagePlayed(object sender,
                          Sanford.Multimedia.Midi.ChannelMessageEventArgs e)
  {
    // послать MIDI-команды звуковой карте
    if(!_MIDIOutDevice.IsDisposed)
        _MIDIOutDevice.Send(e.Message);
  }
}
```

Пример 10.18. Класс MIDIPlayback (версия на VB)

```
Public Class MIDIPlayback
  Implements IPlayback
  Private _MIDISequencer As Midi.Sequencer
  Private _MIDISequence As Midi.Sequence
  Private _MIDIOutDevice As Midi.OutputDevice

  Public Sub New()
    _MIDISequencer = New Midi.Sequencer()
    _MIDISequence = New Midi.Sequence()

    ' обработчики различных событий
    AddHandler _MIDISequencer.ChannelMessagePlayed, _
                  AddressOf sequencer_ChannelMessagePlayed
    AddHandler _MIDISequencer.Chased, AddressOf sequencer_Chased
    AddHandler _MIDISequencer.Stopped, AddressOf sequencer_Stopped
  End Sub

  Public Sub Start() Implements IPlayback.Start
    _MIDISequencer.Start()
  End Sub

  Public Sub Load(ByVal seq As Sequence) Implements IPlayback.Load
    ' найти первое MIDI-устройство
    If _MIDIOutDevice Is Nothing Then
        _MIDIOutDevice = New Midi.OutputDevice(0)
    End If

    ' загрузить MIDI-файл
    _MIDISequence.Load(seq.MusicFile)
    _MIDISequencer.Sequence = _MIDISequence
```

```
    End Sub

    Public Sub Unload() Implements IPlayback.Unload
      If _MIDISequence IsNot Nothing Then
          _MIDISequence.Clear()
      End If

      If _MIDIOutDevice IsNot Nothing Then
          _MIDIOutDevice.Close()
      End If
    End Sub

    Public Sub [Stop]() Implements IPlayback.Stop
        _MIDISequencer.Stop()
    End Sub

    Private Sub sequencer_Stopped(ByVal sender As Object, _
                                  ByVal e As StoppedEventArgs)
      ' послать сообщение "stop" звуковой карте
      For Each message As ChannelMessage In e.Messages
        If (Not _MIDIOutDevice.IsDisposed) Then
            _MIDIOutDevice.Send(message)
        End If
      Next message
    End Sub

    Private Sub sequencer_Chased(ByVal sender As Object, _
                           ByVal e As Sanford.Multimedia.Midi.ChasedEventArgs)
      ' послать сообщение "chased" звуковой карте
      For Each message As ChannelMessage In e.Messages
        If (Not _MIDIOutDevice.IsDisposed) Then
            _MIDIOutDevice.Send(message)
        End If
      Next message
    End Sub

    Private Sub sequencer_ChannelMessagePlayed(ByVal sender As Object, _
                  ByVal e As Sanford.Multimedia.Midi.ChannelMessageEventArgs)
      ' послать MIDI-команды звуковой карте
      If (Not _MIDIOutDevice.IsDisposed) Then
          _MIDIOutDevice.Send(e.Message)
      End If
    End Sub
  End Class
```

Здесь мы настраиваем аппаратное MIDI-устройство, устанавливаем обработчики событий, соответствующих различным сообщениям MIDI, и посылаем эти команды звуковой карте в реальном масштабе времени.

Запись данных

Итак, мы умеем воспроизводить музыку, загружать, сохранять и отображать последовательности. Теперь нужно научиться эти последовательности создавать! При нажатии на кнопку Record в главной форме открывается новая форма. Стоит нажать на кнопку Start recording, как начнется обратный отсчет времени. При счете 0 создается новый поток и начинает проигрываться музыка. Теперь можно создавать последовательность, выстукивая ритм клавишами, соответствующими отдельным каналам платы Phidget Interface Kit. Нажатие клавиши включает канал, отпускание выключает. А, стало быть, лампочки будут загораться и гаснуть.

Чтобы реализовать намеченную программу, нужно подготовить раскладку клавиатуры, то есть установить соответствие между клавишами и каналами. Для этой цели подойдет объект `Dictionary`, отображающий перечисление `Keys` на целочисленный номер канала (см. примеры 10.19 и 10.20).

Пример 10.19. Подготовка раскладки клавиатуры (версия на C#)

```
private Dictionary<Keys, int> _keyMap = new Dictionary<Keys,int>();

public RecordForm()
{
  InitializeComponent();

  // подготовить раскладку клавиатуры
  _keyMap.Add(Keys.D1, 0);
  _keyMap.Add(Keys.D2, 1);
  _keyMap.Add(Keys.D3, 2);
  _keyMap.Add(Keys.D4, 3);
  _keyMap.Add(Keys.D5, 4);
  _keyMap.Add(Keys.D6, 5);
  _keyMap.Add(Keys.D7, 6);
  _keyMap.Add(Keys.D8, 7);
  _keyMap.Add(Keys.D9, 8);
  ...
  _keyMap.Add(Keys.OemQuestion, 39);
}
```

Пример 10.20. Подготовка раскладки клавиатуры (версия на VB)

```
Private _keyMap As Dictionary(Of Keys, Integer) = _
                                    New Dictionary(Of Keys,Integer)()

Public Sub New()
  InitializeComponent()

  ' подготовить раскладку клавиатуры
  _keyMap.Add(Keys.D1, 0)
  _keyMap.Add(Keys.D2, 1)
  _keyMap.Add(Keys.D3, 2)
  _keyMap.Add(Keys.D4, 3)
```

```
    _keyMap.Add(Keys.D5, 4)
    _keyMap.Add(Keys.D6, 5)
    _keyMap.Add(Keys.D7, 6)
    _keyMap.Add(Keys.D8, 7)
    _keyMap.Add(Keys.D9, 8)
    ...
    _keyMap.Add(Keys.OemQuestion, 39)
  End Sub
```

Установив соответствие между клавишами и каналами, мы теперь должны научиться записывать нажатия клавиш через определенные интервалы времени. Сначала напишем обработчики событий `KeyDown` и `KeyUp`, чтобы знать, какая клавиша нажата. С помощью внутреннего массива мы будем сопоставлять каналу булевское значение. Если клавиша нажата, то каналу, соответствующему этой клавише, сопоставляется true, если отпущена – false.

Кроме того, во время звучания песни будет работать отдельный поток. В нем текущие значения из вышеупомянутого массива переписываются в данные канала для текущего отрезка песни. Мы уже отмечали выше, что программа разбивает протяженность песни на отрезки по 50 миллисекунд (один тик). Один раз в 50 мс состояние клавиши помещается в список, ассоциированный с каналом, а счетчик тиков увеличивается на единицу. Все это показано в примерах 10.21 и 10.22.

Пример 10.21. Запись последовательности нажатий клавиш (версия на C#)

```
// список всех каналов в последовательности
private List<Channel> _channels;

// данные о нажатии клавиш будут переписываться отсюда позже
private List<bool[]> _tempData = new List<bool[]>();

private Dictionary<Keys, int> _keyMap = new Dictionary<Keys,int>();
private Sequence _sequence;

// состояние текущего канала
private bool[] _on;

// индекс массива tempData
private int _tickCount = 0;

// песня еще не кончилась?
private bool _playing;

// MCIPlayback или MIDIPlayback
private IPlayback _playback;

private void RecordThread()
{
  float last = 0;
  Stopwatch s = new Stopwatch();

  _playback.Load(_sequence);
```

```
    s.Start();
    _playback.Start();

    _playing = true;

    while(_playing)
    {
      // если прошло достаточно времени, обработать клавиши
      if((s.ElapsedMilliseconds - last) > _sequence.Interval)
      {
        // если дошли до конца, выходим из цикла
        if(_tickCount >= _channels[0].Data.Length)
            btnStart.PerformClick();

        // для каждого канала устанавливаем состояние в этом
        // тике в соответствии с состоянием клавиши
        for(int i = 0; i < _channels.Count; i++)
        {
          if(_on[i])
            _tempData[i][_tickCount] = _on[i];

          // при каждом тике установить состояние выходного
          // порта для текущего канала: вкл или выкл
          if(chkPlay.Checked && !_on[i])
            PhidgetHandler.IFKits[_channels[i].SerialNumber]
              .outputs[_channels[i].OutputIndex] =
                _channels[i].Data[_tickCount];
        }
        _tickCount++;

        // запомнить, когда сделали это в последний раз
        last = (s.ElapsedMilliseconds -
               ((s.ElapsedMilliseconds - last) - _sequence.Interval));
      }
    }
    _playback.Stop();
  }
```

Пример 10.22. Запись последовательности нажатий клавиш (версия на VB)

```
' список всех каналов в последовательности
Private _channels As List(Of Channel)

' данные о нажатии клавиш будут переписываться отсюда позже
Private _tempData As List(Of Boolean()) = New List(Of Boolean())()

Private _keyMap As Dictionary(Of Keys, Integer) = New Dictionary(Of
Keys,Integer)()
Private _sequence As Sequence
Private _thread As Thread

' состояние текущего канала
Private _on() As Boolean

' индекс массива tempData
Private _tickCount As Integer = 0
```

```
' песня еще не кончилась?
Private _playing As Boolean

' MCIPlayback или MIDIPlayback
Private _playback As IPlayback

Private Sub RecordThread()
  Dim last As Single = 0
  Dim s As New Stopwatch()

  _playback.Load(_sequence)

  s.Start()
  _playback.Start()

  _playing = True

  Do While _playing
    ' если прошло достаточно времени, обработать клавиши
    If (s.ElapsedMilliseconds - last) > _sequence.Interval Then
      ' если дошли до конца, выходим из цикла
      If _tickCount >= _channels(0).Data.Length Then
        btnStart.PerformClick()
      End If

      ' для каждого канала устанавливаем состояние в этом
      ' тике в соответствии с состоянием клавиши
      For i As Integer = 0 To _channels.Count - 1
        If _on(i) Then
          _tempData(i)(_tickCount) = _on(i)
        End If

        ' при каждом тике установить состояние выходного
        ' порта для текущего канала: вкл или выкл
        If chkPlay.Checked AndAlso (Not _on(i)) Then
           PhidgetHandler.IFKits(_channels(i).SerialNumber). _
               outputs(_channels(i).OutputIndex) = _
                       _channels(i).Data(_tickCount)
        End If
      Next i
      _tickCount += 1

      ' запомнить, когда сделали это в последний раз
      last = (s.ElapsedMilliseconds - _
            ((s.ElapsedMilliseconds - last) - _sequence.Interval))
    End If
  Loop

  _playback.Stop()
End Sub
```

Если после остановки записи пользователь захочет сохранить записанные данные, то из массива `_tempData` **временные данные будут переписаны в настоящую последовательность, как показано в примерах 10.23 и 10.24.**

Пример 10.23. Копирование данных из временного массива в последовательность (версия на C#)

```
for(int i = 0; i < _tempData.Count; i++)
{
  for(int j = 0; j < _tempData[i].Length; j++)
  {
    if(_tempData[i][j])
       _channels[i].Data[j] = _tempData[i][j];
  }
}
```

Пример 10.24. Копирование данных из временного массива в последовательность (версия на VB)

```
For i As Integer = 0 To _tempData.Count - 1
  For j As Integer = 0 To _tempData(i).Length - 1
    If _tempData(i)(j) Then
       _channels(i).Data(j) = _tempData(i)(j)
    End If
  Next j
Next i
```

В случае MIDI-файла мы можем использовать находящиеся в нем данные для автоматического построения последовательности, которая станет основой окончательного шоу. MIDI-файл содержит серию сообщений звуковой аппаратуре о том, что делать. Эти сообщения можно перебрать, найти среди них те, что представляют для нас интерес, и, взяв из них временные метки, построить нашу последовательность.

Воспроизведение последовательности

Код воспроизведения последовательности очень похож на код ее записи. Чтобы начать воспроизведение песни, используется подходящий класс. Как и раньше, мы создаем отдельный поток, который отсчитывает интервалы по 50 мс. Как только очередной интервал истек, поток проверяет, нужно ли в этот момент включить или выключить канал, и посылает соответствующее значение в выходной порт интерфейсной платы. Код приведен в примерах 10.25 и 10.26.

Пример 10.25. Воспроизведение последовательности (версия на C#)

```
private void ThreadHandler()
{
  float last = 0;
  _tickCount = 0;

  _playback.Load(_sequence);

  _stopWatch = new Stopwatch();
  _stopWatch.Start();

  _playback.Start();
```

```
    _playing = true;

    while(_playing)
    {
      // интервал истек?
      if((_stopWatch.ElapsedMilliseconds - last) >= _sequence.Interval)
      {
        // проверить, кончилась ли песня
        if(_tickCount >= _sequence.Channels[0].Data.Length)
        {
          // выйти и дать знать подписчикам, что все кончилось
          Thread.Sleep(100);
          _stopWatch.Stop();
          _playback.Stop();
          OnSequenceStopped(new EventArgs());
          return;
        }

        // при каждом тике осуществляем запись в выходной порт
        // текущего канала: вкл или выкл
        foreach(Channel c in _sequence.Channels)
        {
        if(PhidgetHandler.IFKits[c.SerialNumber].outputs[c.OutputIndex] !=
               c.Data[_tickCount])
        PhidgetHandler.IFKits[c.SerialNumber].outputs[c.OutputIndex] =
               c.Data[_tickCount];
        }

        _tickCount++;
        last = (_stopWatch.ElapsedMilliseconds -
              ((_stopWatch.ElapsedMilliseconds - last) - _sequence.Interval));
      }
      else
        // пусть ЦП немножко отдохнет
        Thread.Sleep((int)(_stopWatch.ElapsedMilliseconds - last));
    }
    Thread.Sleep(100);
    _stopWatch.Stop();
    _playback.Stop();
  }
```

Пример 10.26. Воспроизведение последовательности (версия на VB)

```
  Private Sub ThreadHandler()
    Dim last As Single = 0
    _tickCount = 0

    _playback.Load(_sequence)

    _stopWatch = New Stopwatch()
    _stopWatch.Start()

    _playback.Start()

    _playing = True
```

```
    Do While _playing
      ' интервал истек?
      If (_stopWatch.ElapsedMilliseconds - last) >= _sequence.Interval Then
        ' проверить, кончилась ли песня
        If _tickCount >= _sequence.Channels(0).Data.Length Then
          ' выйти и дать знать подписчикам, что все кончилось
          Thread.Sleep(100)
          _stopWatch.Stop()
          _playback.Stop()
          OnSequenceStopped(New EventArgs())
          Return
        End If

        ' при каждом тике осуществляем запись в выходной порт
        ' текущего канала: вкл или выкл
        For Each c As Channel In _sequence.Channels
           If PhidgetHandler.IFKits(c.SerialNumber).outputs(c.OutputIndex) <> _
                  c.Data(_tickCount) Then
              PhidgetHandler.IFKits(c.SerialNumber).outputs(c.OutputIndex) = _
                  c.Data(_tickCount)
          End If
        Next c

        _tickCount += 1
        last = (_stopWatch.ElapsedMilliseconds - _
               ((_stopWatch.ElapsedMilliseconds - last) - _sequence.Interval))
      Else
        ' пусть ЦП немножко отдохнет
        Thread.Sleep(CInt(Fix(_stopWatch.ElapsedMilliseconds - last)))
      End If
    Loop
    Thread.Sleep(100)
    _stopWatch.Stop()
    _playback.Stop()
  End Sub
```

Создание шоу

Итак, оборудование собрано, программа написана – самое время создать последовательность зажигания лампочек.

Прежде всего, проверьте, что установлены библиотеки Phidget API. Затем включите платы Phidget в USB-порты своего компьютера. В этот момент аппаратуре может быть еще не придан «товарный вид», и гирлянды подключать необязательно.

Запустите приложение Light Sequencer и выберите из меню пункт File→New→Sequence. Появится окно, показанное на рис. 10.13.

В диалоговом окне выберите файл типа WAV, MP3 или MIDI. Затем введите название, исполнителя и продолжительность песни.

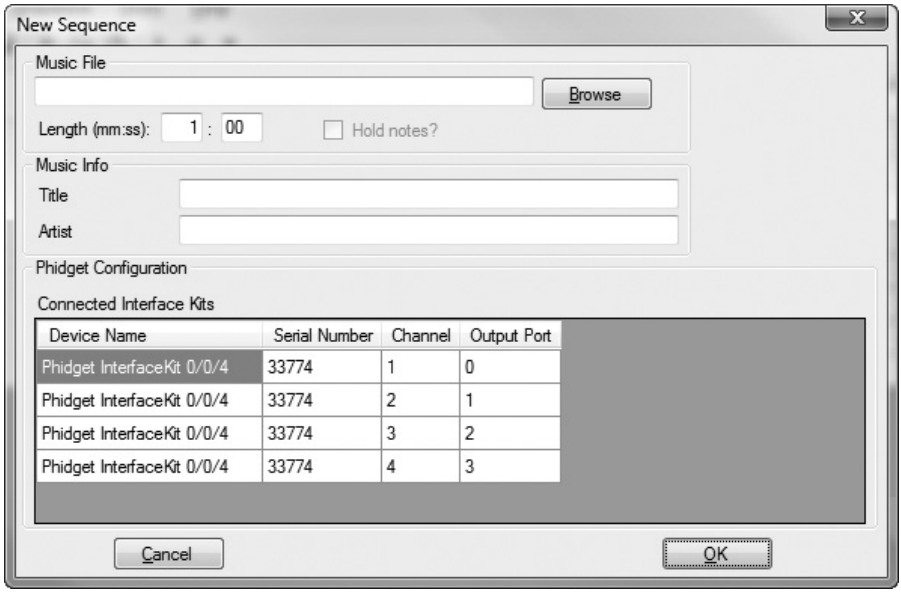

Рис. 10.13. *Диалоговое окно New Sequence*

В сетке, находящейся в нижней части окна, отображаются все подключенные к компьютеру интерфейсные платы Phidget. В любой момент вы можете удалить или добавить новую плату.

В столбце Output Port установите соответствие между выходным портом платы и номером канала, который будет отображаться в сетке последовательности. Можете просто оставить значения по умолчанию, если только нет особых причин отобразить конкретный порт на конкретный канал.

Если вы указали MIDI-файл, то появится дополнительный столбец MIDI Channel. Он содержит выпадающие списки, в которых перечислены все MIDI-каналы в загруженном файле. Обычно один канал отображается на один инструмент и соответствующим образом маркируется, но это необязательно. В этом столбце укажите канал, который вы хотите отобразить на выходной порт платы Phidget (он, в свою очередь, отображается на канал в сетке последовательности).

Нажмите OK. Появится сетка последовательности, которую мы уже раньше видели на рис. 10.12. Если вы загрузили файл типа WAV или MP3, то сетка будет пустой. В случае MIDI-файла сетка будет заполнена данными, извлеченными из самого файла.

Чтобы начать создание светошоу для загруженной песни, выберите из меню пункт Sequence→Record Sequence или нажмите кнопку Record. Появится новое окно, изображенное на рис. 10.14.

Рис. 10.14. *Окно записи последовательности*

Сначала укажите, как помещать записываемые данные в сетку. Если вы хотите затереть уже имеющиеся данные, то выберите вариант Overwrite channel data, а если добавить в конец записанных ранее, то Append channel data. Обычно вы дописываете данные в конец по мере построения последовательности, а первый вариант имеет смысл выбирать только, если вы хотите начать все сначала.

Когда соберетесь отбивать ритм, нажмите кнопку Start. В окне начнется обратный отсчет от 3 до 0, чтобы дать вам время на подготовку, а потом зазвучит песня. Отбивайте на клавиатуре ритм для каждого канала, отмечая, когда включать, а когда выключать гирлянды. Раскладка клавиатуры приведена в табл. 10.2.

Таблица 10.2. Соответствие между клавишами и каналами

Клавиши	Каналы
1–0 (верхняя строка)	1–10
Q–P (вторая строка)	11–19
A–; (третья строка)	20–29
Z–/ (четвертая строка)	30–39

Закончив запись, нажмите кнопку Stop или клавишу Esc. Не считайте, что вы должны за один присест записать последовательность для всех каналов. Как раз для этого и предназначена функция дозаписи. Например, на канале 1 вы можете отбить ритм фонового сопровождения. Покончив с этим, начните запись сначала и отбейте ритм еще для одного-двух каналов. Так постепенно и добавляйте данные. Кстати, флажок Play existing channel data позволяет воспроизводить данные одновременно с записью канала, чтобы можно было видеть, как вспыхивают

лампочки, *в режиме реального времени по мере записи дополнительных каналов*.

После нажатия кнопки Stop программа спросит, хотите ли вы дописать данные в сетку. Если вы ответите Yes, то данные будут добавлены, в противном случае все, что вы только что настучали, будет отброшено, и вы вернетесь в окно записи.

В любой момент вы можете воспроизвести последовательность, выбрав из меню пункт Sequence→Play Sequence или нажав кнопку Play. Программа будет одновременно проигрывать песню и управлять интерфейсными платами, включая и выключая каналы в соответствии с тем, что вы записали.

Есть и другой способ взаимодействия с последовательностью – с помощью самой главной сетки. Отдельные ячейки можно включать или выключать, щелкая по ним мышью или буксируя мышь сразу по группе ячеек. Кроме того, можно включить квадратик клавишей «O» и выключить клавишей «F». Наконец, щелчок по ячейке правой кнопкой мыши открывает контекстное меню с командами On и Off (рис. 10.15).

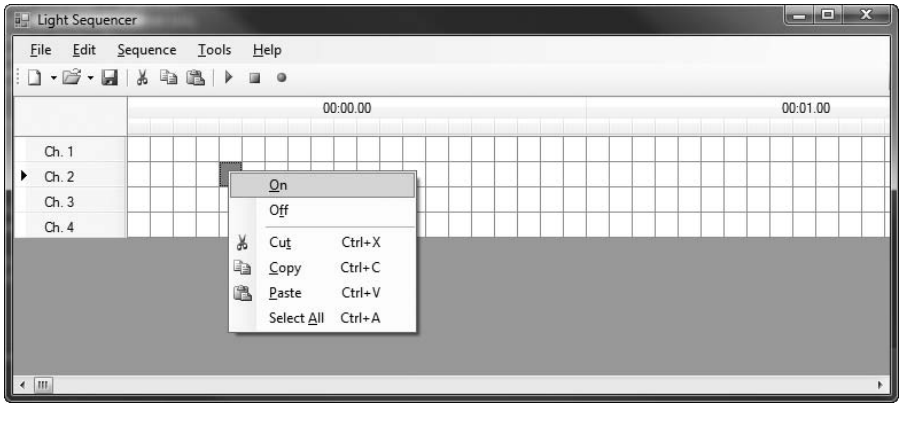

Рис. 10.15. *Включение и выключение ячеек*

Так можно вводить данные с нуля или стирать существующие данные, если ритмическая последовательность оказалась слишком короткой или слишком длинной.

Можно также вырезать, копировать и вставлять группы ячеек. Это очень удобно, когда нужно повторить часть последовательности для нескольких отрезков песни. Для этого выделите диапазон ячеек и выберите из меню Edit команду Cut или Copy. Затем выделите ячейку, начиная с которой хотите выполнить вставку, и выберите из меню Edit команду Paste.

И последняя функция программы – создание списков воспроизведения. Если вы хотите записать последовательности для нескольких песен, а потом воспроизвести их подряд, то можете создать из последовательностей список воспроизведения.

Выберите из меню команду File→New→New Playlist. Появится окно, показанное на рис. 10.16.

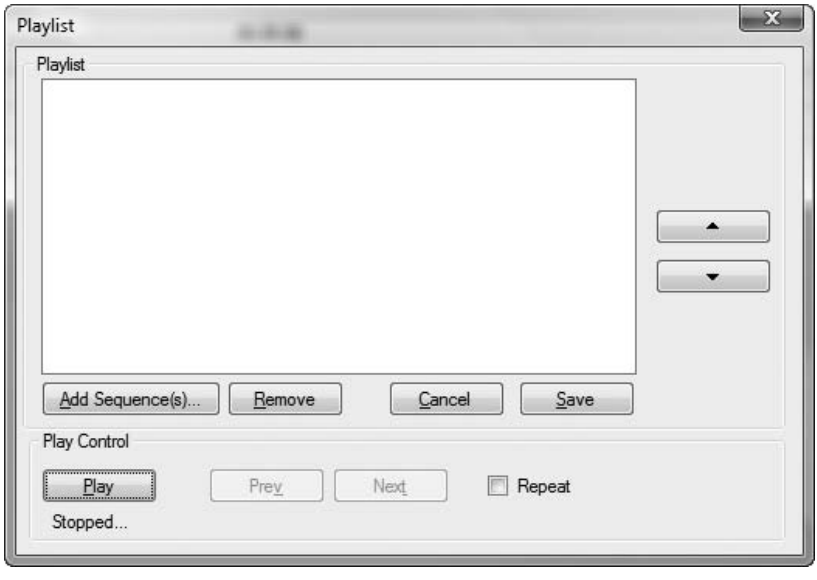

Рис. 10.16. *Окно создания списка воспроизведения*

Чтобы добавить в список новую последовательность, нажмите кнопку Add Sequence. Появится стандартное окно поиска файлов, в котором вы сможете указать одну или несколько добавляемых последовательностей. По завершении операции добавления можете с помощью кнопок со стрелками справа от списка переупорядочить последовательности в нем.

Закончив формирование списка, нажмите кнопку Save, чтобы сохранить его в файле.

Для воспроизведения списка нажмите кнопку Play, и все перечисленные в нем песни будут воспроизведены в указанном порядке. Отметьте флажок Repeat, если хотите повторять воспроизведение в цикле, пока не прервете вручную.

Заключительные замечания

Вот мы и сделали это: несложное оборудование и удобная программа, позволяющая создать праздничное светошоу. Или будничное, если на то пошло. Эта программа использовалась для оформления световых шоу, музейных экспозиций с применением робототехники и даже для управления поющим роботом в развлекательном центре Chuck E. Cheese![1]

Нам было бы интересно знать, какие последовательности придумаете вы. Пожалуйста, свяжитесь с нами через сайт книги *http://www.c4fbook.com/*, мы поможем продвинуть ваше творчество в массы!

[1] Сеть семейных развлекательных центров в США. – *Примеч. перев.*

A

Использование файла C4fStyle в WPF-проектах

Обзор файла C4fStyle

В проектах InnerTube и PeerCast, созданных с помощью технологии Windows Presentation Foundation (WPF), используется набор стилей, определенных в файле *C4fStyle.xaml*. Вы можете применить их и в собственных WPF-приложениях.

Включение файла C4fStyle в приложение

Чтобы воспользоваться файлом *C4fStyle.xaml*, вы должны сначала добавить в свой проект ссылку на сборку *PresentationFramework.Aero.dll*, как показано на рис. A.1. Это необходимо для использования стиля `Button`, который зависит от стиля `ButtonChrome`, определенного в библиотеке Aero Presentation Framework.

Затем включите файл *C4fStyle.xaml* в свой проект в Visual Studio. Чтобы применить находящиеся в этом файле стили на уровне приложения (это будет означать, что стили всех кнопок наследуют *C4fStyles.xaml*), мы просто включаем показанную в примере A.1 XAML-разметку в файл *App.xaml* (в VB-проектах он называется *Application.xaml*).

Пример A.1. Ссылка на словарь ресурсов C4fStyles

```
<Application.Resources>
  <ResourceDictionary>
    <ResourceDictionary.MergedDictionaries>
      <ResourceDictionary Source="C4fStyles.xaml"/>
    </ResourceDictionary.MergedDictionaries>
  </ResourceDictionary>
</Application.Resources>
```

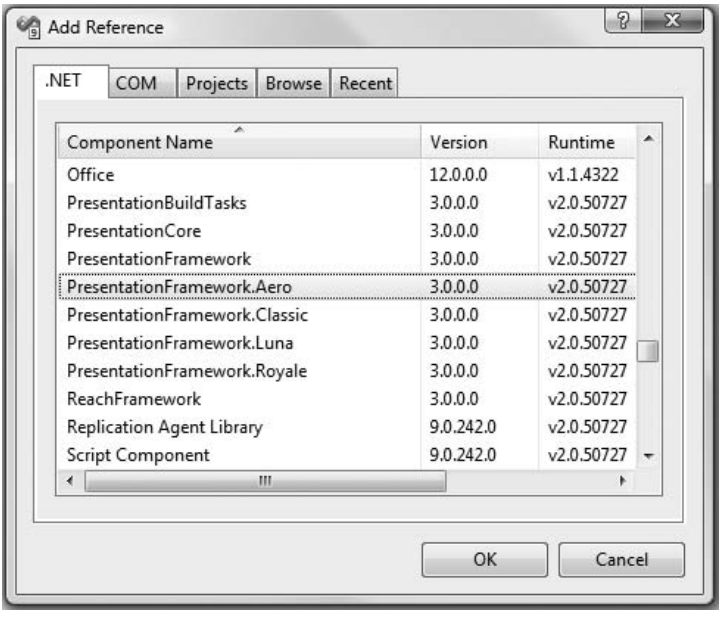

Рис. А.1. *Добавление ссылки на библиотеку Aero Presentation Framework*

Общие и именованные стили

В файле *C4fStyles.xaml* определены стили двух видов: общие и именованные. Общие стили автоматически изменяют внешний вид всех кнопок, меток и списков, определенных в текущей области видимости. Например, если установить общий стиль кнопки на уровне приложения, как показано в примере А.1, то изменятся стили всех кнопок, встречающихся в этом приложении.

Примечание

Внешний вид элементов управления, унаследовавших общий стиль, можно изменить вручную, установив свойство `Style`. Явное задание свойства `Style` отменяет общий стиль, заданный на уровне приложения.

Напротив, для использования именованного стиля необходимо явно установить свойство `Style` элемента управления, как показано в примере А.2.

Пример А.2. Присваивание именованного стиля свойству HeaderStyle

```
<Label Name="MyLabel" Style="{DynamicResource HeaderStyle}" >Hello</Label>
```

Справка по стилям

В табл. А.1 и А.2 приведена информация об общих и именованных стилях, определенных в файле *C4fStyles.xaml*.

Таблица А.1. Общие стили в файле C4fStyles.xaml

Элемент управления	Описание
`Label`	Все метки выводятся полужирным шрифтом черного цвета.
`Button`	Для вывода кнопок применяется темно-серый градиент с анимацией по событию щелчка мышью.
`ListBox`	Изменяет внешний вид границ, цвет, полосы прокрутки и другие свойства списков.
`ListBoxItem`	Изменяет внешний вид, цвет, фон и другие свойства элементов списков.

Таблица А.2. Именованные стили в файле C4fStyles.xaml

Имя	Элемент управления	Область применения	Описание
`Label`	Label	Элементы управления Label	Крупный полужирный шрифт, оранжевый цвет
`SilverGradient`	Нет	Фон окна	Радиально-градиентная кисть серебристого цвета
`Orange`	Нет	HeaderStyle	Оранжевый цвет
`AnimateMediaList`	ListBox	ListBox	Анимирует раскрытие и свертывание списка

До и после

С помощью файла *C4fSstyle.xaml* мы можем изменить подразумеваемый по умолчанию внешний вид интерфейса, изображенный на рис. А.2, так что он будет выглядеть, как показано на рис. А.3.

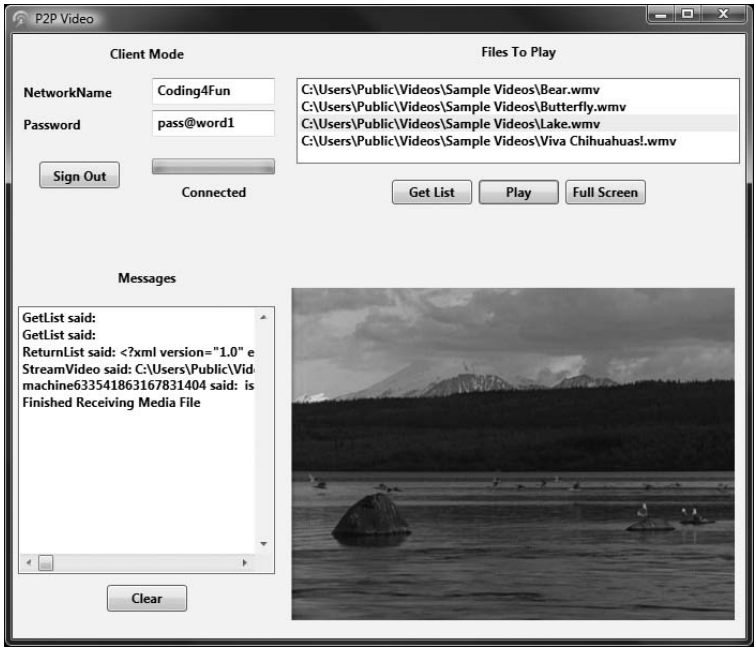

Рис. А.2. *Приложение PeerCast со стилями по умолчанию*

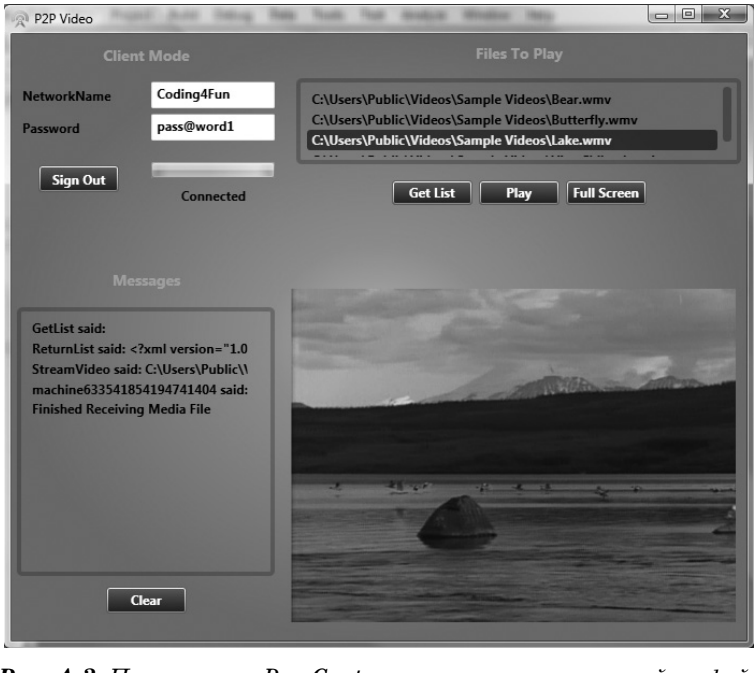

Рис. А.3. *Приложение PeerCast после применения стилей из файла C4fSstyle.xaml*

Алфавитный указатель

G

H

I

J

K

L

M

R

S

T

U

V

W

X

Y

Т

У, Ф

Ч

Ш

Э

Я