БИБЛИОТЕКА ПРОГРАММИСТА

Microsoft

Д. Эспозито Ф. Эспозито

РАЗРАБОТКА ПРИЛОЖЕНИЙ для Windows 8 на HTML5 и JavaScript

- Работа с Microsoft Visual Studio 2012 Express
- Основные сведения о HTML, CSS и JavaScript
- Пошаговые инструкции по разработке приложения
- Взаимодействие с датчиками и аксессуарами устройств
- Публикация приложения в Windows Store

ПИТЕР

Microsoft

Start Here!™

Build Windows 8 Apps with HTML5 and JavaScript

Dino Esposito

Francesco Esposito

БИБЛИОТЕКА ПРОГРАММИСТА

Д. Эспозито, Ф. Эспозито

РАЗРАБОТКА ПРИЛОЖЕНИЙ для Windows 8 на HTML5 и JavaScript

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

Дино Эспозито, Франческо Эспозито

Разработка приложений для Windows 8 на HTML5 и JavaScript

Перевел с английского Н. Вильчинский

Заведующий редакцией	*А. Кривцов*
Руководитель проекта	*А. Юрченко*
Ведущий редактор	*Ю. Сергиенко*
Литературный редактор	*А. Жданов*
Художественный редактор	*Л. Адуевская*
Корректор	*И. Тимофеева*
Верстка	*Л. Родионова*

ББК 32.988.02-018
УДК 004.438.5

Эспозито Д., Эспозито Ф.

Э85 Разработка приложений для Windows 8 на HTML5 и JavaScript. — СПб.: Питер, 2014. — 384 с.: ил. — (Серия «Библиотека программиста»).

ISBN 978-5-496-00794-8

С помощью этой книги вы быстро освоите разработку приложений для Windows 8 с использованием таких технологий, как HTML5 и JavaScript. Написанное известным экспертом Дино Эспозито в соавторстве со своим сыном, это практическое пособие содержит все необходимое для того, чтобы помочь читателю спроектировать, создать и опубликовать свое приложение для Windows 8.

Издание состоит из трех частей. В первой части рассматриваются вопросы использования Microsoft Visual Studio 2012 Express, а также даются краткие сведения об HTML, CSS и JavaScript. Во второй части книги рассматриваются основы программирования для Windows 8 с предоставлением пошаговых упражнений, помогающих освоить пользовательский интерфейс Windows 8, графику, видео, хранилища данных, интернет-вызовы. В третьей части основное внимание уделяется современному программированию для Windows 8 с упором на работу с датчиками и аксессуарами устройств (такими как принтеры, GPS, веб-камеры и т. д.), взаимодействию с системой и публикации готового приложения в Windows Store.

12+ (Для детей старше 12 лет. В соответствии с Федеральным законом от 29 декабря 2010 г. № 436-ФЗ.)

ISBN 978-0735675940 англ.

ISBN 978-5-496-00794-8

ООО «Питер Пресс», 192102, Санкт-Петербург, ул. Андреевская (д. Волкова), д. 3, литер А, пом. 7Н.
Налоговая льгота — общероссийский классификатор продукции ОК 005-93, том 2; 95 3005 — литература учебная.
Подписано в печать 01.11.13. Формат 70x100/16. Усл. п. л. 30,960. Тираж 2000. Заказ
Отпечатано в полном соответствии с качеством предоставленных издательством материалов в ОАО «Тверской ордена Трудового Красного Знамени полиграфкомбинат детской литературы им. 50-летия СССР».
170040, Тверь, проспект 50 лет Октября, 46.

Содержание

Посвящается Микеле и Сильвии, которые намного сильнее и умнее, чем они сами о себе думают.

Дино

Посвящается бабушке Кончите за бесплатные калории в виде восхитительной домашней ветчины в невероятных количествах.

Франческо

Благодарности

Дино:

Буду честен: взяться за этот проект меня уговорил мой редактор в O'Reilly Media Рассел Джонс (Russell Jones). Если вы держите в руках эту книгу, то все ее достоинства и недостатки следует отнести и на его счет! Когда Рассел впервые завел речь о книге еще на уровне простого замысла, я отказался, мотивируя это тем, что никогда не писал книг для начинающих.

Но потом мой сын Франческо (вполне достойный и умелый соавтор) заставил меня взглянуть на проект с другой точки зрения. Он продемонстрировал одну из тех эффектных форм нестандартного мышления, которые иногда свойственны молодым. Франческо сказал примерно следующее: «Папа, я не думаю, что твоя работа как программиста или твои классы для ASP.NET MVC интересны только зрелым специалистам. Если бы я был одним из них, меня вряд ли заинтересовал бы какой-нибудь класс, а если он меня все-таки заинтересует, мне понадобится, чтобы кто-то рассказал мне о нем не спеша и доходчиво. А если я решаю вложить деньги в изучение класса, значит, в чем-то считаю себя новичком. Почему же к книге нужно относиться по-другому?»

И этот посыл дошел до меня, я признал его мысль здравой, хотя она и исходила от четырнадцатилетнего парня.

Итак, изменив свои взгляды, я с энтузиазмом взялся за этот проект и попросил Франческо работать над ним вместе со мной, поскольку он отлично справлялся с тестированием, питаясь, по сути, той пищей для ума, которую мы сами готовили! Франческо проделал действительно выдающуюся работу. Бывало, я уже находился на борту готового к взлету самолета и по телефону советовал ему, как усовершенствовать фотогалерею или загрузить с Flickr данные в формате JSON.

Стороннему наблюдателю этот звонок представлялся не чем иным, как классическим деловым звонком, одним из тех, которые делаются в последнюю перед взлетом самолета минуту. Но я-то разговаривал с собственным сыном! И, что еще важнее, к моему возвращению все задания были им старательно выполнены. Спасибо, Франческо!

Франческо:

Мне нравятся технологии, платформа и инструментарий разработки от Microsoft. Поначалу для меня написание книги по большей части было способом приложить руки к устройству под названием Surface. Но в конечном счете большую часть времени я провел за работой с симулятором и вспомогательным ноутбуком.

Отец сказал, что изучение новой технологии обычно дается нелегко, потому что при этом нельзя полагаться на документацию или на доступность хороших примеров. Честно говоря, для меня это звучало как одно из тех оправданий, к которым прибегают родители, когда они не могут что-то сделать сами. Не задумываясь о возможных трудностях, я просто засучил рукава и разработал ряд примеров. И в процессе этой работы я сумел увеличить свой вклад в проект, высказав отцу ряд замечаний. И не думаю, что без моего участия этот проект стал бы для моего отца таким приятным.

Хотя к работе над книгой я относился главным образом как к развлечению, признаю, что книга стала для меня важным достижением. Я знаю, что буду чувствовать себя еще лучше, если смогу разделить свой успех с теми, кто сделал мою жизнь счастливее: с моей мамой, сестрой Микелой, моими друзьями, Франческо и Маттиа, с моими товарищами по ватерпольной команде UISP Monterotondo. Я всех вас люблю!

PS: Микела, ты помнишь Рождество 2009 года, когда я так сильно доставал тебя, что отец решил меня отвлечь и попробовать заинтересовать (и вправду заинтересовал?) программированием?

Об авторах

Дино Эспозито, уже много лет работающий инструктором и высококлассным консультантом, является автором многих популярных книг издательства Microsoft Press, которые помогли профессиональному росту тысяч разработчиков и архитекторов программного обеспечения на платформе .NET. Дино является техническим директором быстрорастущей компании, предоставляющей программное обеспечение и мобильные сервисы профессиональным спортсменам, а теперь он еще и технический последователь JetBrains, где занимается разработкой для Android и Kotlin, а также входит в команду управления WURFL — базой данных мобильных устройств, используемых такими организациями, как Google and Facebook. Следите за сообщениями Дино на твиттере по адресу @despos и на сайте http://software2cents.wordpress.com.

Франческо Эспозито, еще только тинэйджер (ему всего лишь 15 лет), но уже накопил весьма богатый опыт в разработке мобильных приложений для различных платформ, включая iOS с Objective C и MonoTouch, Android с Java, Windows Phone, и даже BlackBerry. Он написал основной объем кода для IBI12 — официального мультиплатформенного приложения для Rome ATP Masters 1000.

Кроме создания приложений он любит общаться с друзьями, заниматься водным поло и ходить в школу, где стремится получать наивысшие баллы, позволяющие достичь его заветной цели — добиться стипендии в Гарварде или просто купить собственный планшетный компьютер Surface.

Введение

Долгие годы программирование было исключительной прерогативой тех, кого считали специалистами экстра-класса, гуру или гениями. Однако с появлением мобильных устройств ситуация изменилась, поскольку шанс писать программы для таких устройств появился даже у молодежи. Сегодня уже нередки ситуации, когда благодаря своему умению создавать интересные приложения для Android или iPhone в центре внимания неожиданно оказывается подросток. Причин тому несколько.

Разумеется, одной из причин является то, что сегодняшние подростки относятся к поколению цифрового мира. При поиске программиста у вас сегодня гораздо больше шансов найти представителя этой группы, нежели представителя предыдущих поколений. Еще одной причиной является то, что приложения для мобильных систем писать гораздо проще, чем другие виды современного программного обеспечения. Мобильные приложения представляют собой небольшие фрагменты кода, выстроенные вокруг какой-нибудь удачной идеи. Одно дело создать мобильное приложение и совсем другое — спланировать и поддерживать многозвенную корпоративную систему.

Мобильность добавила к разработке программного обеспечения новое измерение. В этом плане Windows Phone — это не просто еще одна мобильная платформа, для которой можно создавать программы, это пока что простейшая (и самая приятная) платформа программирования независимо от того, что положено в ее основу. Благодаря этому программирование для Windows Phone является отличной стартовой площадкой для начинающих программистов. В особенности это касается тех смышленых подростков, которые постоянно охотятся за новыми технологиями и радуются их появлению. Я видел все это на примере собственного 14-летнего сына Франческо, который в результате стал соавтором данной книги.

С появлением Microsoft Windows 8 «программирование с удовольствием» получило еще один импульс. Windows 8 позволяет создавать не только мобильные приложения для планшетных устройств, но и полноценные приложения для собственного применения, призванные просто развлекать или автоматизировать некоторые рутинные операции в повседневной жизни. Благодаря своей относительной простоте, операционная система Windows 8 вернула приложениям некую размерность искусства, которая была утрачена за последнее десятилетие или около того из-за усложнения архитектуры программного обеспечения и веб-сайтов. С одной стороны, Windows 8 является эффективной клиентской частью многофункционального и сложного программного обеспечения промежуточного слоя, с другой стороны, эта операционная система достаточно проста для любого, кто решит заняться программированием с использованием HTML5 и JavaScript.

Эта книга может рассматриваться в качестве краткого (но интересного) руководства для тех, кто приступает к освоению искусства создания приложений для Windows 8 и постигает секреты публикации и продажи этих приложений через Магазин Windows. Главное в этой книге — дать вам понять, что при наличии хорошей идеи и способности к быстрому обучению вы без проблем сможете создать Windows-приложение независимо от своего возраста или опыта программирования. Вы узнаете, как создавать работоспособные приложения для новой операционной системы компании Microsoft, как заставить их работать на настольных и планшетных компьютерах. В качестве доказательства не могу не отметить, что хотя мой сын еще подросток, он уже написал большинство примеров и несколько глав для этой книги.

Прочитав книгу, вы не станете специалистом экстра-класса, но для написания собственных приложений знаний вам точно хватит.

Кому нужно читать эту книгу

Это руководство ориентировано на начинающих программировать для Windows 8 с использованием таких веб-технологий, как HTML5 и JavaScript. Однако значение слова «начинающих» требует дополнительного разъяснения. Одно из определений относится к тем, кто никогда всерьез не обучался программированию. Хотя данная книга и ориентирована на таких новичков, она все же требует некоторых минимальных знаний о HTML5 и JavaScript, а также знакомства с основными понятиями логики и формализма, такими как операции IF, WHILE и присваивание. Еще одно определение «начинающих» подразумевает тех, кто никогда не изучал программирование для Windows, но кто, возможно, десятилетиями писал программы на Коболе или последние 15 лет создавал и сопровождал приложения, созданные на Visual Basic 6. Хотя эта книга может стать полезной и для этих более опытных «начинающих», те, кто имеет солидный опыт программирования, целевой аудиторией для нее не являются.

В этой книге мы постарались подойти к ключевым темам программирования для Windows 8 достаточно плавно. Если вам интересна именно система Windows 8 и не приходилось сталкиваться с Windows Phone, Microsoft Silverlight или даже с одностраничными приложениями, то вам обязательно нужно подумать о приобретении данной книги.

Кому не нужно читать эту книгу

Эта книга не сделает из вас первоклассного разработчика программ для Windows 8. При наличии солидного опыта работы с системой Windows 8, Windows Phone или Silverlight или с другими языками программирования, можно взять более сложную книгу или довольствоваться онлайновыми MSDN-документами либо StackOverflow-ссылками. Чтобы книга понравилась, вы реально должны быть новичком в Windows 8.

Организация книги

Книга состоит из трех тематических частей. В главах с 1-й по 5-ю рассматриваются вопросы получения и использования Microsoft Visual Studio 2012 Express, а также даются краткие сведения о HTML, CSS и JavaScript. В главах с 6-й по 11-ю речь идет о создании приложений для Windows 8 и рассматриваются основы программирования для Windows 8 с предоставлением пошаговых упражнений, помогающих освоить пользовательский интерфейс Windows 8, графику, видео, хранилища данных, интернет-вызовы. И наконец, в главах с 12-й по 14-ю основное внимание уделяется современному программированию для Windows 8 с упором на работу с датчиками и аксессуарами устройств (такими, как принтеры, GPS, веб-камеры и т. д.), взаимодействию с системой (живыми плитками) и публикацию готового приложения.

С чего начинать

В целом сценарий использования этой книги довольно прост. Мы рекомендуем прочесть ее от корки до корки, поскольку она скомпонована таким образом, чтобы познакомить вас с ключевыми темами программирования для Windows 8 средствами HTML5 и JavaScript. Но если вы уже хорошо разбираетесь в технологиях Visual Studio 2012 Express, HTML5, CSS и JavaScript, можете пропустить главы с 1-й по 4-ю, не опасаясь, что это помешает вам разобраться в остальных главах книги.

Принятые соглашения

Информация в книге преподносится с соблюдением некоторых соглашений, призванных сделать книгу понятнее и проще в плане усвоения материала.

- Каждое упражнение состоит из ряда заданий, которые нужно выполнить в рамках упражнения.
- Врезки предоставляют дополнительную информацию или альтернативные методы для успешного завершения того или иного этапа.
- Текст, который нужно набирать, а также блоки кода представлены моноширинным шрифтом.
- Знак плюс (+) между двумя названиями клавиш означает, что эти клавиши нужно нажимать одновременно. Например, фраза «Нажмите Alt+Tab» означает, что вам нужно нажать клавишу Tab, удерживая нажатой клавишу Alt.
- Значок ▸ между двумя или несколькими командами, например File (Файл) ▸ Close (Закрыть), означает, что нужно раскрыть меню и выбрать нужную команду или, если таких значков несколько, сначала раскрыть подменю, а уже затем выбрать нужную команду, и т. д.

Системные требования

Для настройки и компиляции примеров кода понадобится персональный компьютер, оснащенный Windows 8 и Visual Studio 2012 Express для Windows 8.

Примеры кода

Большинство глав этой книги выстроено вокруг упражнений, которые отражены в примерах кода для той или иной главы. Все учебные проекты в своем завершенном виде могут быть загружены со страницы http://aka.ms/SH_W8AppsHTML5JS/files.

Там же вы найдете инструкции для загрузки файла starthere-buildapps-winjs-sources.zip.

Установка примеров кода

Для установки примеров кода на свой компьютер, чтобы их можно было использовать в упражнениях, выполните следующие действия:

1. Разархивируйте файл starthere-buildapps-winjs-sources.zip, загруженный с веб-сайта книги (если нужно, укажите в пути распаковки конкретный каталог).
2. Когда появится приглашение, прочитайте выведенное на экран лицензионное соглашение с конечным пользователем. Если вас все устраивает, примите соглашение и щелкните на кнопке Next (Далее).

ПРИМЕЧАНИЕ

Если лицензионное соглашение не появится, его можно посмотреть на той же веб-странице, с которой вы загрузили файл starthere-buildapps-winjs-sources.zip.

Список опечаток и поддержка книги

Мы прилагаем все усилия, чтобы не допустить неточностей в этой книге и сопутствующем контенте. Все ошибки, замеченные после издания книги, перечислены в разделе Microsoft Press сайта oreilly.com по адресу http://aka.ms/SH_W8AppsHTML5JS/errata.

При обнаружении ошибки, не попавшей в список, о ней можно сообщить на той же веб-странице.

Если нужна дополнительная поддержка, отправьте сообщение в отдел Microsoft Press Book Support по адресу mspinput@microsoft.com.

Пожалуйста, имейте в виду, что поддержка программных продуктов компании Microsoft по этому адресу не предоставляется.

От издательства

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

Мы будем рады узнать ваше мнение!

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

1 Visual Studio 2012 express edition для Windows 8

Если цели совпадают, то различия в обычаях и языке никакого значения не имеют.

Дж.К. Роулинг, «Гарри Поттер и кубок огня»

Выход Microsoft Windows 8 ознаменовал дебют существенно переработанной платформы выполнения для Windows (Windows RunTime, WinRT). Так же как и .NET, платформа WinRT поддерживает несколько языков программирования. Вас ожидает приятный сюрприз (и старый знакомый) — наряду с популярными языками, использующимися в .NET (такими, как C#, Visual Basic, C++, F#), поддерживается и JavaScript.

ПРИМЕЧАНИЕ

Вы можете даже не вспомнить, что десять лет назад, когда компания Microsoft впервые выпустила .NET Framework, разработчикам предоставлялась возможность создавать приложения с использованием адаптированной версии JavaScript, которая называлась JScript .NET. Но успеха это начинание не снискало; а сейчас, конечно же, в ведущей среде разработки .NET-кода, Visual Studio, поддержку JScript .NET вы не найдете. Десять лет назад язык JavaScript в плане своей популярности был близок к самой низкой отметке. А то, что JScript .NET был диалектом стандартного языка JavaScript, не означало, что вы можете задействовать технологии HTML и CSS для формирования пользовательского интерфейса в ходе выполнения приложения. Однако с выходом Windows 8 ситуация изменилась.

Создание приложений для Windows 8 с применением JavaScript означает, что макет пользовательского интерфейса создается средствами HTML, а стилевое оформление и графика добавляются средствами CSS. С точки зрения логики работы приложения используется стандартный язык JavaScript, обогащенный любыми JavaScript-библиотеками на ваше усмотрение (например, широко распространенной библиотекой jQuery), а доступ к системным WinRT-классам осуществляется посредством специально для этого предназначенной и созданной компанией Microsoft JavaScript-оболочкой — библиотекой WinJs.

Если вы уже сталкивались с JavaScript-разработкой, создание приложений для Windows 8 не превратится для вас в некое совершенно неизвестное приключение. Если же заниматься JavaScript-разработкой вам еще не приходилось, маршрут, пройденный при изучении JavaScript, станет для вас, возможно, кратчайшим путем к освоению навыков создания приложений для Windows 8.

В данной главе делаются предварительные наметки этого пути и рассматриваются все вопросы, касающиеся установки конкретно Windows 8 и Microsoft Visual Studio, а также настройки среды. В следующих главах сначала будут даны краткие сведения о HTML (в частности, о самой последней версии HTML, известной как HTML5), CSS и JavaScript, а затем мы попробуем создать приложение для Windows 8, попутно изучая вопросы, имеющие непосредственное отношение к программированию для Windows 8.

ВНИМАНИЕ

Если вам уже знакомы технологии HTML5, CSS и JavaScript, то можете сразу приступать к изучению главы 5, а если нет, я, как минимум, рекомендую вам внимательно изучить главы 2, 3 и 4. Еще лучше будет заглянуть в специализированные книги по HTML5 и JavaScript, поскольку в представленных здесь главах содержится не более 10 % от объема специальных книг. Можно изучить и другие книги серии Microsoft Press, в которых рассматриваются именно эти темы: «Start Here! Learn HTML5», Faithe Wempen, (Microsoft Press, 2012) и «Start Here! Learn JavaScript», Steve Suehring, (Microsoft Press, 2012).

Подготовка к разработке приложений

Итак, у вас есть желание приступить к созданию приложений для Windows 8 с использованием HTML, CSS и JavaScript. Сначала нужно убедиться в том, что на предназначенной для разработки машине должным образом установлено необходимое программное обеспечение.

Необходимое программное обеспечение

Очевидно, чтобы разрабатывать, тестировать и запускать приложения для Windows 8, нужно установить Windows 8 на компьютере. Проще всего разрабатывать и тестировать приложения для Windows, используя текущую версию Visual Studio —Visual Studio 2012.

Существует множество редакций как Windows 8, так и Visual Studio 2012, но для работы с данной книгой достаточно минимальных версий каждого продукта, каковыми являются Windows 8 Basic edition и свободно распространяемая версия Visual Studio 2012 Express edition для разработки Windows-приложений.

Установка Windows 8

Основным предварительным условием работы с информацией и упражнениями данной книги является оснащение машины операционной системой Windows 8, которая представлена несколькими разновидностями (табл. 1.1).

Таблица 1.1. Редакции Windows 8

Версия	Описание
Windows 8	Редакция Windows 8 Basic доступна для обеих архитектур: x86 и x86-64. В ней представлен новый начальный экран и переделанный пользовательский интерфейс, живые плитки, Internet Explorer 10 и многое другое
Windows 8 Pro	Эта редакция предлагает дополнительные возможности, такие как начальная загрузка с виртуального жесткого диска и поддержка виртуализации посредством Hyper-V
Windows 8 Enterprise	Эта редакция добавляет такие связанные с IT-технологиями возможности, как AppLocker и Windows-To-Go (начальная загрузка и работа с USB-накопителя). Она также поддерживает установку приложений собственной разработки с мест, отличных от Магазина Windows (Windows Store)
Windows 8 RT	Доступна только предустановленной на планшетных компьютерах на базе ARM, также изначально включает в себя оптимизированные под сенсорное управление версии основных приложений Office 2013

Если у вас еще нет собственной копии Windows 8, по адресу http://msdn.microsoft.com/en-us/evalcenter/jj554510.aspx можно получить бесплатную пробную 90-дневную версию. Учтите, что по этой ссылке вы получите необновляемую копию Windows 8 Enterprise. Перед тем как приступать к загрузке, нужно иметь в виду, что она занимает несколько гигабайтов, поэтому загрузка может занять много времени!

Установка Visual Studio express

После установки Windows 8 можно приступать к загрузке Visual Studio 2012 Express edition. (Учтите, что далее в этой главе да и во всей книге используется название Visual Studio или Visual Studio 2012, которое, как правило, означает Visual Studio 2012 Express edition.) Как показано в табл. 1.2, среда Visual Studio доступна в различных видах.

Таблица 1.2. Редакции Visual Studio 2012

Версия	Описание
Ultimate	Полноценная версия Visual Studio 2012, предлагающая самую качественную поддержку каждой функциональной возможности
Premium	Отсутствуют некоторые расширения в области моделирования, отладки и тестирования
Professional	Отсутствуют почти все функциональные возможности, касающиеся моделирования, отладки и тестирования, тем не менее предлагается вполне подходящая среда для написания и тестирования кода
Express	Свободная, но минимальная версия Visual Studio 2012, оптимизированная для специфических сценариев разработки. В частности, в ней можно создавать веб-приложения или приложения для Windows 8

Более развернутая сравнительная характеристика версий Visual Studio есть на странице http://www.microsoft.com/visualstudio/11/en-us/products/compare.

Чтобы приступить к загрузке Visual Studio Express for Windows 8, перейдите в раздел Dev Center for Windows 8 applications на веб-сайте http://msdn.microsoft.com/en-us/windows/apps (рис. 1.1).

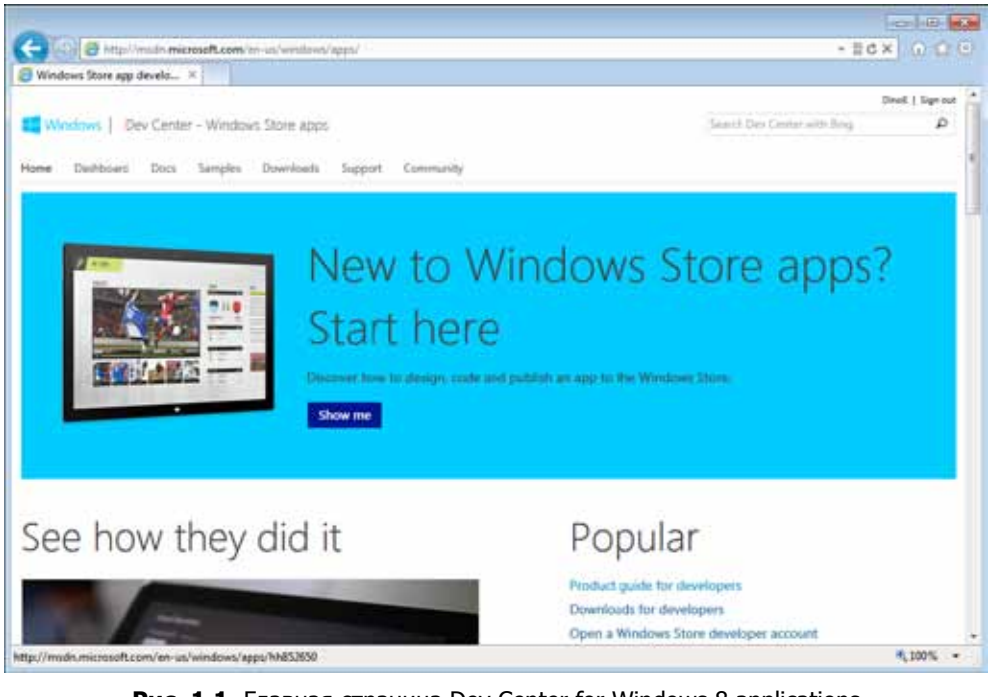

Рис. 1.1. Главная страница Dev Center for Windows 8 applications

После щелчка на ссылке, предназначенной для перехода к загрузке инструментов и набора для разработки программного обеспечения (Software Development Kit, SDK), вы попадете на другую страницу, где можно наконец-то приступить к загрузке (рис. 1.2).

Рис. 1.2. Загрузка инструментальных средств разработки приложений для Windows 8

Рис. 1.3. Среда Visual Studio 2012 готова к запуску

Можно выбрать либо сохранение программы установки на вашем локальном диске, либо ее непосредственный запуск. Если вы собираетесь многократно использовать программу на различных машинах, то лучше ее сначала сохранить в известном вам месте.

В процессе установки периодически будут выводиться приглашения на принятие или изменение параметров. Для работы с этой книгой можно просто принимать все параметры, предлагаемые по умолчанию. Это позволит установить самую новую среду .NET Framework 4.5, Windows 8 SDK, плюс группу других инструментальных средств и шаблонов проектов. Если все пройдет гладко, то в конце установки появится экран, воспроизведенный на рис. 1.3. В случае неудачи при установке программного обеспечения будет выдано сообщение с рядом полезных инструкций, которые подлежат неукоснительному выполнению.

Теперь можно приступить к настоящей работе: запуску и настройке Visual Studio, что позволит вам создать ваше первое приложение для Windows 8!

Настройка Visual Studio 2012

После завершения установки Visual Studio 2012 Express требует выполнения ряда подготовительных действий.

Получение ключа продукта

Во время первого запуска Visual Studio 2012 требует активации своей копии, которая осуществляется с помощью экрана, показанного на рис. 1.4.

Рис. 1.4. Для запуска Visual Studio 2012 Express требуется ключ продукта

Щелчок на ссылке Register online (Регистрация на сайте) приведет вас на страницу, где можно ввести свои имя, адрес электронной почты и сведения о компании (рис. 1.5).

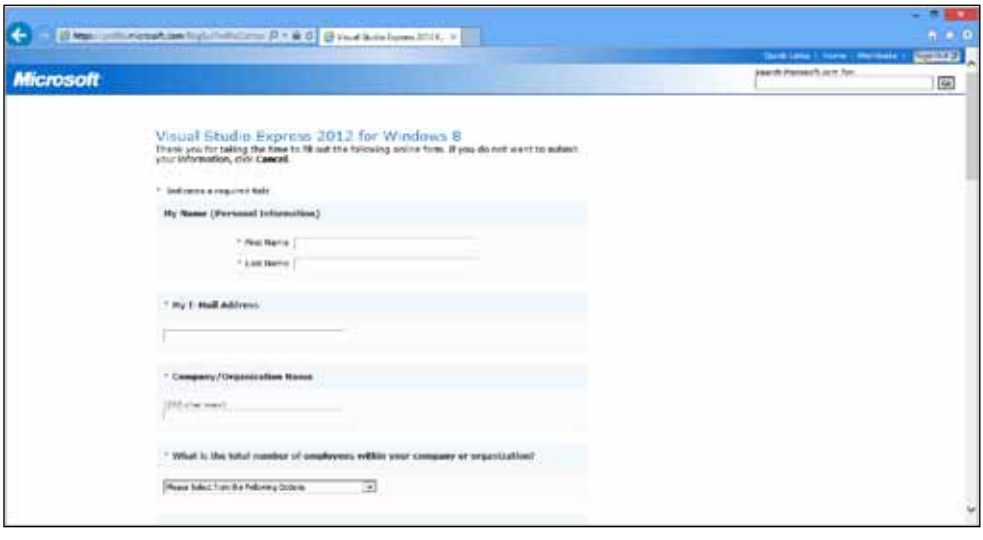

Рис. 1.5. Для работы с Visual Studio 2012 Express требуется регистрация

После ввода данных нужно отправить форму и получить сообщение по электронной почте, содержащее ключ продукта (рис. 1.6).

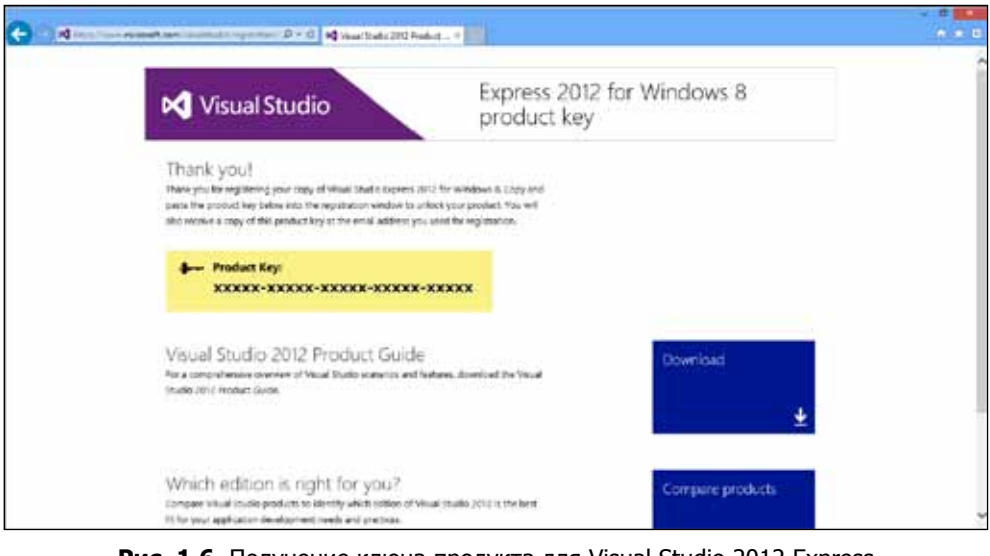

Рис. 1.6. Получение ключа продукта для Visual Studio 2012 Express

Получение сообщения электронной почты от компании Microsoft на указанный вами адрес занимает всего несколько секунд. В этом сообщении в текстовой форме содержится ключ продукта, который нужно скопировать в буфер обмена и перейти обратно в окно Visual Studio. Вставьте только что полученный ключ в окно, показанное на рис. 1.4.

Создание учетной записи разработчика

Чтобы создавать и тестировать приложения для Windows 8, нужно получить от компании Microsoft лицензию разработчика. Лицензия предоставляется бесплатно и дает вам право считаться зарегистрированным разработчиком Microsoft. Как показано на рис. 1.7, получение такой лицензии требует только подписи с использованием вашего Windows-идентификатора. Если у вас нет такого идентификатора, в диалоговом окне, предлагающем его ввести, можно перейти по ссылке Sign up (Зарегистрироваться).

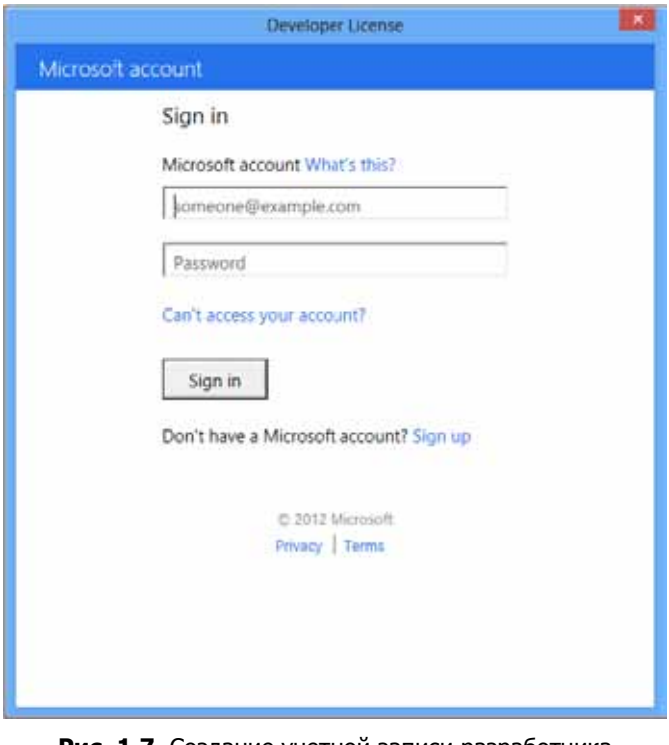

Рис. 1.7. Создание учетной записи разработчика

Лицензия разработчика, успешно установленная на машине, позволяет свободно создавать и запускать приложения для Windows 8 вне официального Магазина Windows.

ПРИМЕЧАНИЕ

На машину с Windows 8 могут устанавливаться только сертифицированные приложения, либо загруженные из Магазина Windows (практически так же, как это делается с приложениями для Windows Phone), либо созданные зарегистрированными разработчиками на «подписанной» машине, поэтому для выполнения примеров, приводимых в книге, вам нужна лицензия разработчика.

Теперь, пока не истечет срок действия лицензии или пока она не будет отозвана с машины, вы больше не будете получать от системы никаких предупреждений. Если срок действия лицензии истечет, ее можно будет обновить непосредственно из среды Visual Studio. Для обновления лицензии пользователи Visual Studio Express раскрывают меню Store (Магазин), а затем, как показано на рис. 1.8, выбирают пункт Acquire Developer License (Приобрести лицензию разработчика).

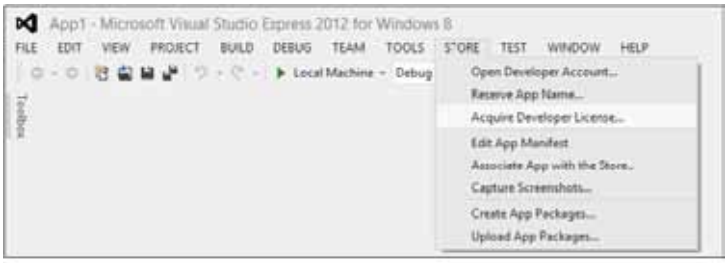

Рис. 1.8. Обновление лицензии разработчика

При условии, что у вас есть учетная запись с Windows-идентификатором, лицензий можно получать сколько угодно.

Учетная запись в Магазине Windows

Получение лицензии разработчика для Windows 8 — это только первый обязательный шаг в разработке и тестировании приложений для Windows 8. Имея лишь учетную запись разработчика, вы не можете публиковать приложения для Windows 8 в Магазине Windows с целью их загрузки и установки другими пользователями.

Прямой связи между учетными записями разработчиков и учетными записями в Магазине Windows не существует. Каждая из них играет свою особенную роль, поэтому можно получить одну, не имея другой. Но если у вас есть учетная запись в Магазине Windows, а затем вы приобрели лицензию разработчика, то срок действия вашей лицензии разработчика автоматически увеличивается.

Важно помнить, что перед получением возможности публикации своих приложений для Windows 8 в Магазине Windows, вам нужно создать учетную запись в Магазине Windows. Как получить учетную запись в Магазине Windows, рассказывается в главе 14.

ВНИМАНИЕ

Вам как разработчику и пользователю системы Windows 8 следует помнить, что на вашей машине можно запускать только те приложения для Windows 8, которые были загружены из Магазина Windows, либо нестандартные приложения, для которых на машине была установлена лицензия разработчика. Есть еще один сценарий, позволяющий запускать нестандартные приложения — эти приложения должны быть «загружены со стороны» на машину вашей организацией, у которой, в свою очередь, имеется в Магазине корпоративная учетная запись.

Начало работы с приложениями для Windows 8

Получив для своей машины, работающей под управлением Windows 8, лицензию разработчика, вы сможете запускать приложения для Windows 8. Перед тем как приступить к разработке нового проекта, нужно выбрать шаблон проекта и язык программирования. После этого Visual Studio поможет вам, сгенерировав простой код, соответствующий шаблону и языку, и этот код можно далее приспосабливать под свои задачи и расширять.

В соответствии с темой книги в качестве языка программирования следует выбрать JavaScript. Но стоит все же напомнить, что вы можете воспользоваться и другими языками, например C#, Visual Basic и даже C++.

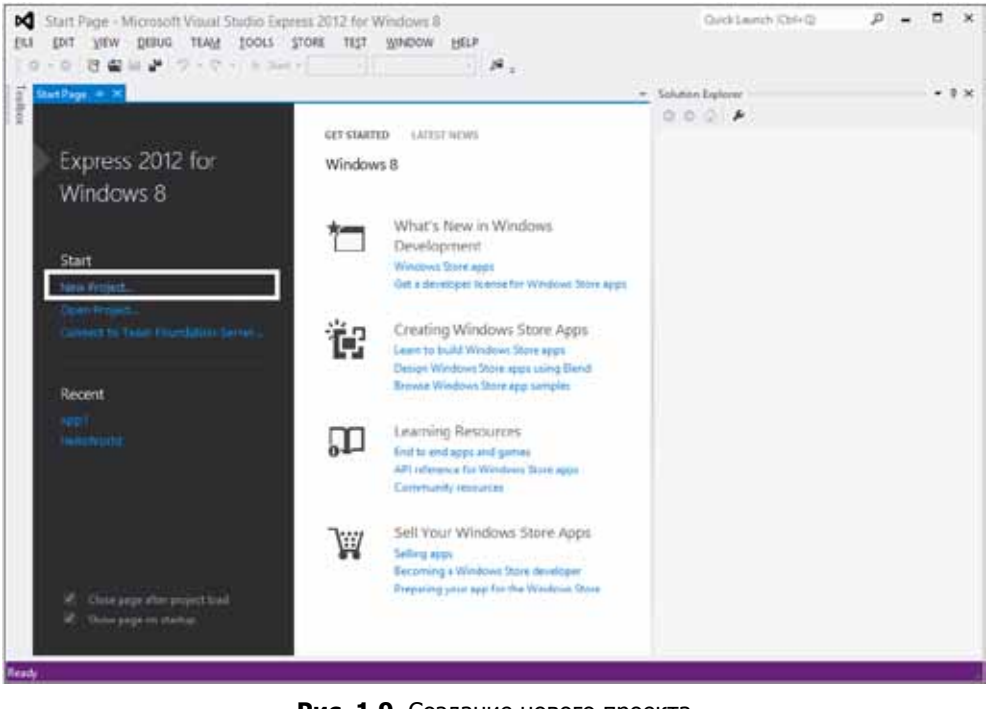

Рис. 1.9. Создание нового проекта

Приложение «Hello Windows 8»

Без дальнейших предисловий запустите Visual Studio и узнайте, что нужно для создания нового проекта. На самом деле все проще простого, на начальной странице, показанной на рис. 1.9, нужно щелкнуть на ссылке New Project (Новый проект).

Выбор шаблона проекта

Для вашего нового проекта Visual Studio предлагает ряд предопределенных шаблонов, однако выбрать шаблон проекта не так-то просто, как это может показаться. Для этого нужно иметь четкое представление о конечном результате. Выбор нужного шаблона зависит от представляемой вами модели взаимодействия графики и контента, с которыми придется работать. Окно New Project (Новый проект), которое появляется после выбора команды создания проекта, показано на рис. 1.10.

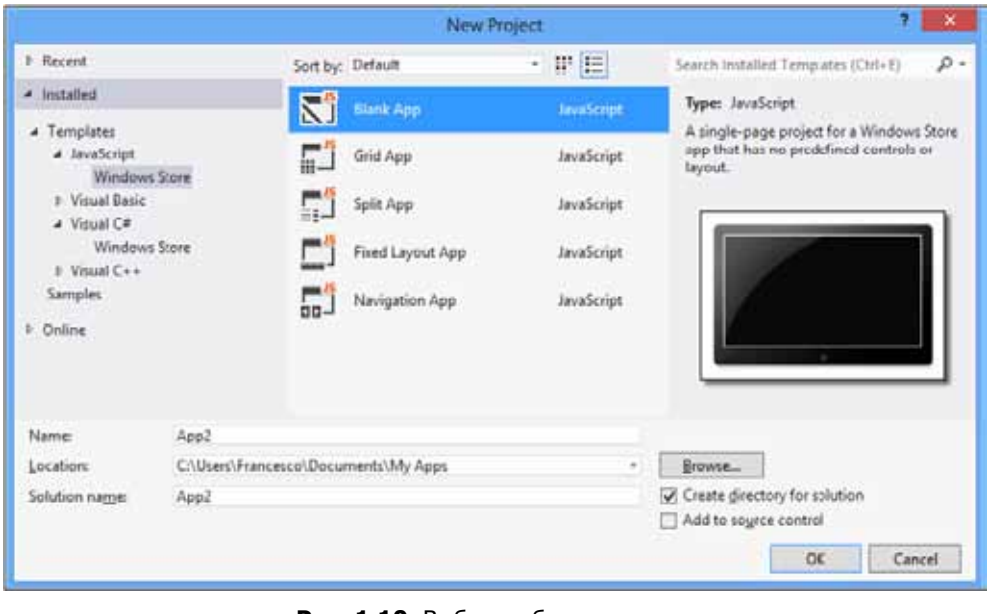

Рис. 1.10. Выбор шаблона проекта

Шаблоны сгруппированы по языкам программирования. В редакции Visual Studio Express, рассматриваемой в данной книге, можно создавать приложения только одного типа — приложения, предназначенные для Магазина Windows. Если приобрести более совершенную редакцию Visual Studio, откроются новые возможности по созданию приложений, включая веб-приложения, консольные приложения и приложения для настольных систем.

Но как же все-таки решить, каким из шаблонов воспользоваться?

Главное назначение шаблонов — сэкономить ваши усилия, по крайней мере, в отношении обычного макета приложения. Но выбирать тот или иной шаблон вас никто не заставляет. Если вы считаете, что ни один из предопределенных шаблонов вам не подходит или, что чаще бывает, вы просто не знаете, какой из них нужно выбрать, просто выберите шаблон приложения, создаваемого с «чистого листа» (вариант Blank App). Сведения о предопределенных шаблонах для JavaScript даны в табл. 1.3.

Таблица 1.3. Предопределенные шаблоны для приложений Магазина Windows

Шаблон	Описание
Blank App	Приложение, состоящее из одной практически пустой страницы: на ней нет ничего определенного, ни видимых элементов управления, ни виджетов, ни макета
Grid App	Трехстраничное приложение со структурой страниц «главная-подчиненные». На главной странице элементы сгруппированы в сетке. Подчиненные страницы предоставляют детали, касающиеся сгруппированных и отдельных элементов
Split App	Двухстраничное приложение со структурой страниц «главная-подчиненная», где на главной странице показаны выбираемые элементы, а на подчиненной странице взаимосвязанные элементы перечисляются рядом друг с другом
Fixed Layout App	Одностраничное приложение, макет которого пропорционально масштабируется
Navigation App	Многостраничное приложение с предопределенными элементами управления для навигации по страницам

Для изучения материалов данной книги проще всего начать с абсолютно нового, пустого приложения. С шаблонами остальных типов вы сможете поэкспериментировать в следующих главах книги.

Создание учебного проекта

Перед тем как дать Visual Studio зеленый свет на создание файлов, следует подумать о том, где разместить проект. На рис. 1.10 показано поле Location (Местоположение), в которое можно ввести путь на диске для файлов, создаваемых в рамках проекта.

Учебные приложения лучше всего сохранять в определенной файловой структуре. Для демонстрационного кода данной книги будет использоваться корневой каталог Win8, содержащий для каждой главы каталоги Ch*XX*, где *XX* — это номер главы, состоящий их двух цифр.

По умолчанию Visual Studio сохраняет файлы вашего проекта непосредственно в папке Documents, создавая для каждого решения новый каталог. Исходное местоположение проекта можно изменить, в любой момент отредактировав путь в поле Location (Местоположение). Новый путь, предлагаемый по умолчанию для каждого проекта, можно также установить, выбрав в меню Tools (Сервис) команду Options (Настройки), а затем выбрав в разделе Projects and Solutions (Проекты и решения) узел General (Общие настройки), как показано на рис. 1.11.

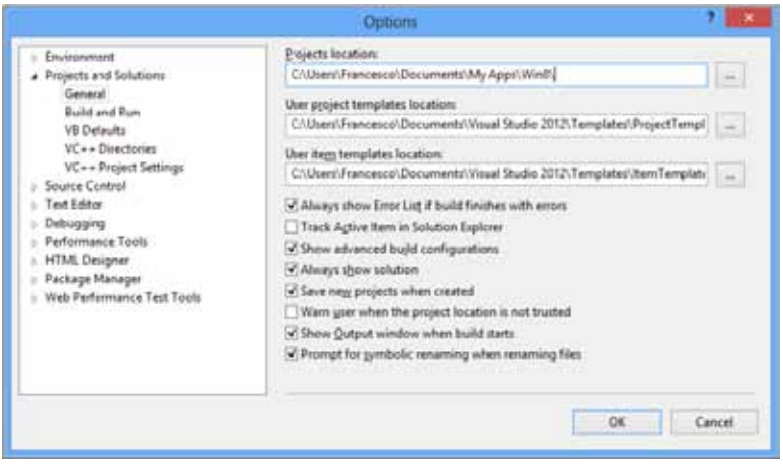

Рис. 1.11. Выбор для проекта места, предлагаемого по умолчанию

Для приложения «Hello Windows 8» в папке Win8/Ch01 нужно создать новый пустой проект по имени HelloWin8 (рис. 1.12).

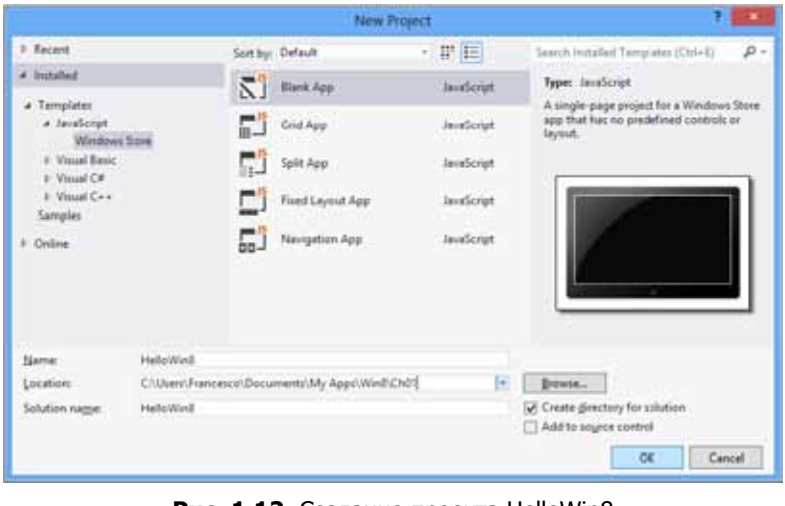

Рис. 1.12. Создание проекта HelloWin8

После щелчка на кнопке OK вы официально становитесь разработчиком приложений для Windows 8.

ПРИМЕЧАНИЕ

По умолчанию Visual Studio 2012 поставляется с темной темой для окон и элементов управления. Чтобы получать более качественные распечатки экрана, эту тему нужно сменить на светлую. Для смены темы Visual Studio нужно выбрать в меню команду Tools (Сервис) ▸ Options (Настройки) и в открывшемся окне, показанном на рис. 1.11, выбрать вариант Environment (Среда).

Внесение изменений в учебный проект

Сразу же после создания проект HelloWin8 выглядит так, как показано на рис. 1.13. В нем содержится ссылка на библиотеку Windows для JavaScript (которая находится в папке References), и он ориентирован на HTML-страницу под названием default.html. На этой странице определен весь пользовательский интерфейс приложения и ссылки на файл css/default.css каскадной таблицы стилей (Cascading Style Sheet, CSS), описывающей графическое оформление, а также на JavaScript-файл js/default.js, задающий логику загрузки контента страницы и любое ожидаемое поведение. Файл default.js открывается в Visual Studio по умолчанию.

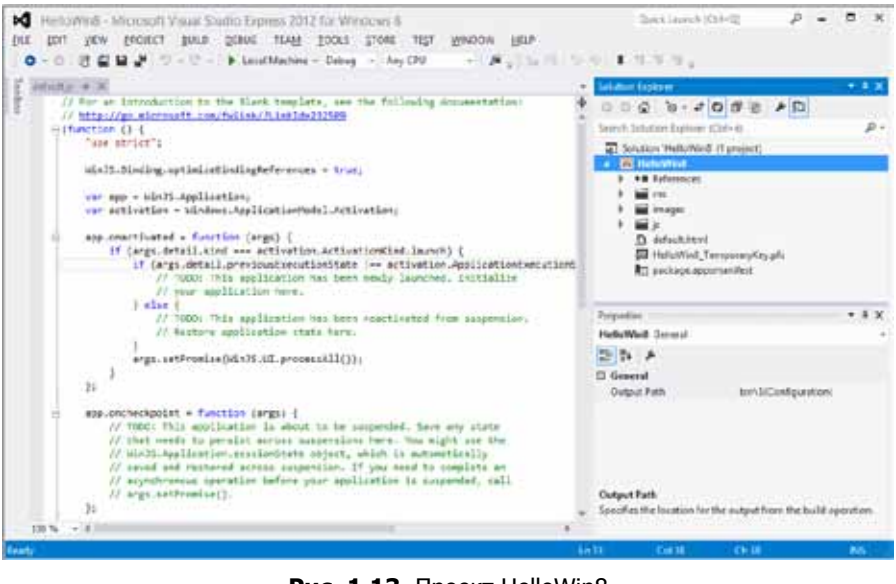

Рис. 1.13. Проект HelloWin8

Оказывается, приложения для Windows 8 записываются на JavaScript-коде, который напоминает код самостоятельного веб-приложения с HTML-страницами,

должным образом стилизованными с помощью CSS-стилей и усиленными JavaScript-логикой. Если вы уже знакомы с концепцией веб-приложений и веб-разработкой на стороне клиента, то вам останется только разобраться в прикладном программном интерфейсе (Application Programming Interface, API), специфичном для Windows 8 и предлагаемом вам в нескольких JavaScript-файлах, на которые можно ссылаться.

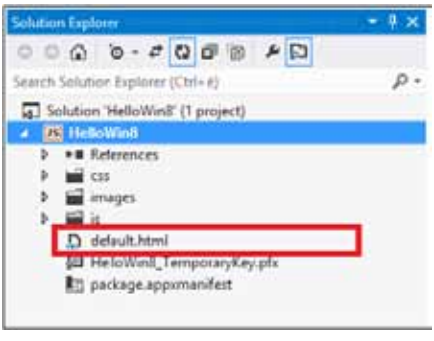

Рис. 1.14. Открытие файла default.html

Перед тем как скомпилировать проект, чтобы посмотреть, что получилось, давайте внесем в него небольшие изменения: закройте файл default.js и откройте файл default.html, который отвечает за главную страницу приложения (рис. 1.14). Чтобы открыть файл, являющийся частью данного проекта, нужно найти его значок на панели Solution Explorer (Проводник решений) и дважды щелкнуть на нем. А если для ссылки нужно открыть файл, не включенный в проект, может понадобиться воспользоваться командой Open (Открыть) в меню File (Файл).

Все тело учебной HTML-страницы имеет следующий вид:

```
<body>
   <p>Content goes here</p>
</body>
```

Давайте просто заменим текст заполнения своим текстом. Например:

```
<body>
   <p> Hello, Windows 8!<p>
</body>
```

В языке HTML элемент `<body>` определяет весь контент страницы. А элемент `<p>` определяет текстовый абзац. В результате внесенного изменения на страницу выводится текст «Hello, Windows 8».

Следующим шагом будет создание приложения и наблюдение за его работой.

Наблюдение за работой приложения

Для создания приложения нужно нажать клавишу F5 или выбрать в меню команду Build (Создать) ▸ Start Debugging (Запустить отладку). Отладка является действием по поиску и устранению ошибок в компьютерных программах. Но команда Build (Создать) ▸ Start Debugging (Запустить отладку) вообще-то относится к пробному запуску приложения. Она позволяет вам запускать приложение и пользоваться им, чтобы сравнивать его поведение с ожидаемым.

Для еще более быстрого запуска можно щелкнуть на кнопке воспроизведения на панели инструментов (рис. 1.15).

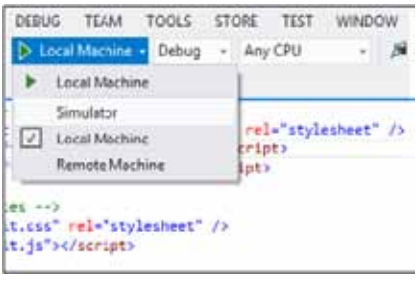

Рис. 1.15. Запуск приложения в режиме отладки

Учтите, что пункт Local Machine (Локальная машина) является лишь вариантом, предлагаемым по умолчанию для запуска приложения. Выбрав этот пункт, вы откроете меню, предлагающее несколько вариантов. Запуск приложения на локальной машине означает переход из режима обычного рабочего стола Windows (в котором выполняется Visual Studio) на специфический пользовательский интерфейс (UI) Windows 8. Если вам это не подходит, приложение можно запустить в симуляторе. Симулятор помогает тестировать приложение с применением разных ориентаций и разрешений экрана. И наконец, приложение можно даже запустить на удаленной машине при условии, что у вас есть соответствующие права на доступ к этой машине.

Меню Debug (Отладка) на рис. 1.15 позволяет выбрать способ создания кода компилятором. В режиме отладки исполняемый файл включает в себя дополнительную информацию, позволяющую устанавливать контрольные точки на определенных строках кода и запускать программу в пошаговом режиме. Контрольной точкой является строка кода, на которой выполнение приостанавливается. Контрольные точки обычно используются для приостановки выполнения кода в заданном месте и исследования состояния приложения и его внутренних данных. В программе может быть несколько контрольных точек. Для завершенных приложений, готовых к распространению, требуется режим выпуска (Release mode). В данной книге будет преимущественно использоваться режим отладки (Debug mode).

На рис. 1.16 показано приложение при запуске на локальной машине.

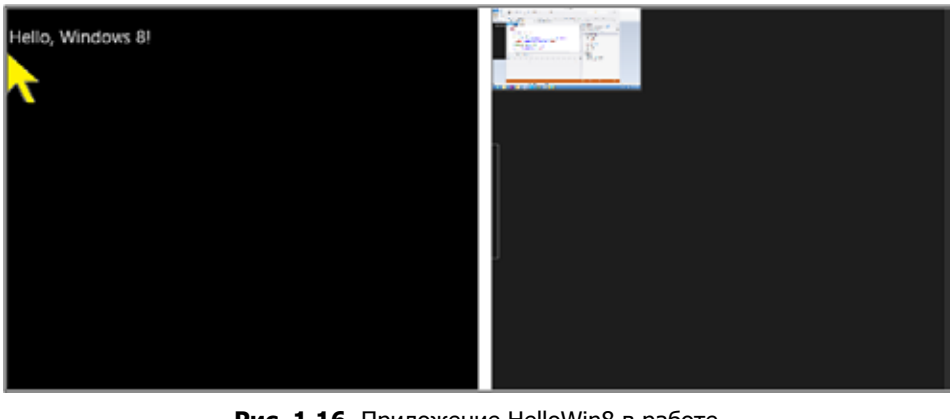

Рис. 1.16. Приложение HelloWin8 в работе

Если приложение запущено в симуляторе, оно работает в отдельном окне, которым при желании можно управлять. При запуске приложения на локальной машине оно занимает весь экран, поэтому сразу непонятно, что нужно делать, чтобы вернуться в Visual Studio и завершить работу приложения. Выход из приложения осуществляется следующим образом: перемещайте указатель мыши в направлении левого края экрана, пока не увидите значок окна, щелчок на котором позволит вернуться в режим рабочего стола. Затем для прекращения работы приложения нужно щелкнуть на кнопке Stop, которая заменяет кнопку пуска в пользовательском интерфейсе Visual Studio.

Вот и все. Не правда ли, просто? А теперь давайте сделаем приложение чуть более привлекательным и чуть более дееспособным.

Добавление новой функциональности

Создайте новый проект и назовите его HelloWin8-Step2. Сначала мы сделаем его немного привлекательнее, добавив HTML-элементы и стилевую информацию, а затем превратим его в простое, но полнофункциональное приложение, генерирующее случайные числа.

Добавление стиля к странице

Откройте файл страницы default.html и измените его тег `body`. Теперь в тело будет включен заголовок и колонтитул, отделенный прямой линией. Для этого мы используем несколько HTML5-элементов (более подробно о HTML5 рассказывается в следующей главе). Измененное тело страницы должно иметь следующий вид:

```
<body>
    <header>
      Start Here! Build <b>Windows 8</b> Applications with <b>HTML5</b> and
      <b>JavaScript</b>
      <hr />
    </header>
    <footer>
      <hr />
      Dino Esposito | Francesco Esposito
    </footer>
</body>
```

А теперь займемся цветовым оформлением и шрифтами. Стиль страницы определен в файле default.css, который находится в папке CSS. Редактируя CSS-файл, можно сделать с внешним видом и макетом HTML-страницы практически все, что угодно. Наиболее важные сведения о CSS есть в главе 3.

Изначально в файле default.css вы найдете примерно следующий код:

```
body {
}
```

Это код описания стиля, применяемого к тегу body любой страницы, ссылающейся на CSS-файл. CSS-файл можно отредактировать вручную или же можно создавать CSS-стили, используя специальное средство, доступное в Visual Studio. Чтобы воспользоваться этим средством, щелкните правой кнопкой мыши на CSS-элементе (то есть на теге) body и выберите пункт Build Style (Создать стиль), как показано на рис. 1.17.

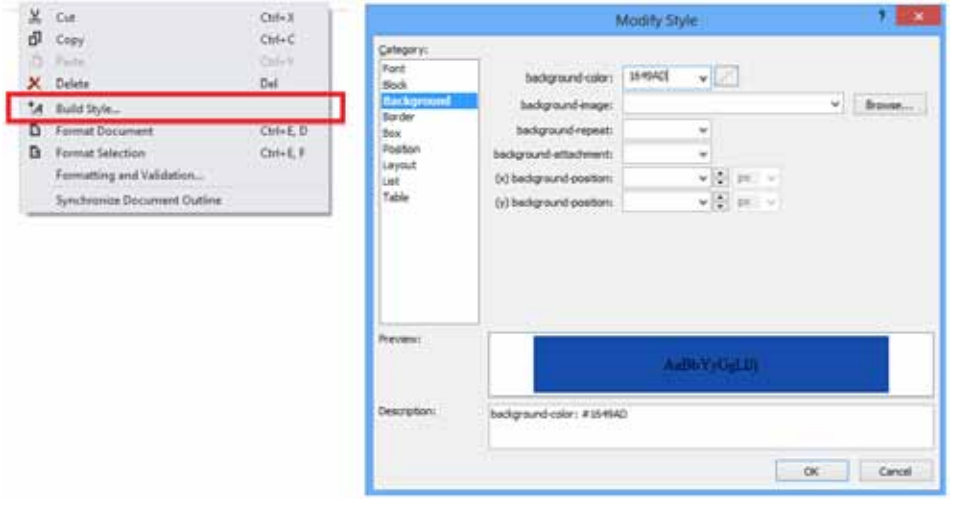

Рис. 1.17. Редактирование стиля страницы

Измененный стиль тела страницы, в котором выбран другой фоновый цвет и добавлено небольшое пространство вокруг контента, должен иметь следующий вид:

```
body {
    background-color: #1649AD;
    padding: 10px;
}
```

Можно также настроить внешний вид элементов заголовка и колонтитула, определив цвет, шрифт и вертикальное смещение текста:

```
header {
    font-size: x-large;
    color: #ffffff;
    padding-bottom: 50px;
 }
 footer {
    font-size: large;
    color: #eeee00;
    padding-top: 50px;
}
```

Теперь запустите приложение и с гордостью посмотрите на результат (рис. 1.18).

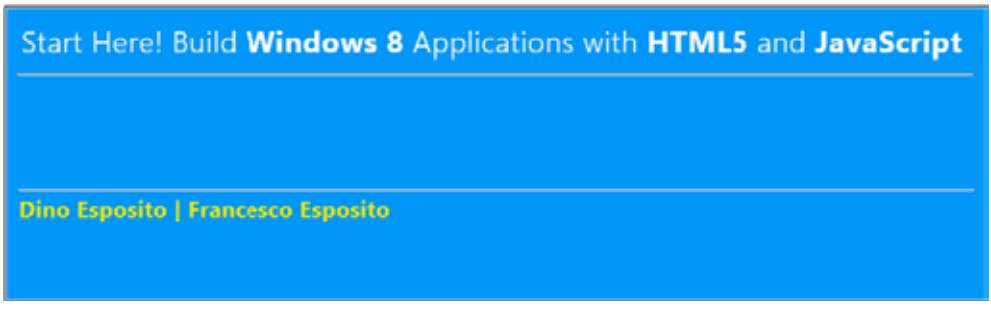

Рис. 1.18. Приложение, запущенное с измененным стилем

Генерирование случайного числа

Пока что приложение вообще никак себя не проявляет, ограничиваясь выводом статического текста. Давайте придадим ему некий смысл и добавим всю необходимую структуру и логику для генерирования и вывода случайного числа.

Первым делом добавим к телу страницы в файле default.html разметку, состоящую из двух блоков `div`, содержащих указатель места для сгенерированного числа и кнопку, на которой нужно будет щелкать для получения нового числа. Вставьте между элементами `header` и `footer` следующую разметку:

```
<div>
    <label id="numberLabel">?</label>
  </div>
  <div>
    <input id="numberButton" type="button" value="Get number" />
</div>
```

Затем откройте файл default.js и добавьте в конце следующие JavaScript-функции:

```
function numberButtonClick() {
    var number = generateNumber();
    document.getElementById("number").innerHTML = number;
  }
  function generateNumber() {
    var number = 1 + Math.floor(Math.random() * 1000);
    return number;
}
```

Первая функция является обработчиком события `click` кнопки. Вторая функция просто генерирует и возвращает случайное число в диапазоне между 1 и 1000. Завершающий шаг состоит в привязке обработчика события `click` к конкретной кнопке в HTML-разметке. Для этого существует несколько способов, простейший из которых реализуется так:

```
<input id="numberButton" type="button" value="Get number"
onclick="numberButtonClick()" />
```

Более изящный и рекомендуемый при программировании для Windows 8 способ заключается в динамической привязке при загрузке страницы. Итак, откройте файл default.js и измените код функции app.onactivated:

```
app.onactivated = function (args) {
   if (args.detail.kind === activation.ActivationKind.launch) {
     if (args.detail.previousExecutionState !==
       activation.ApplicationExecutionState.terminated) {
         // ЗАДАЧА: Это приложение только что было запущено.
         // Инициализируйте здесь ваше приложение
         document.getElementById("numberButton").addEventListener(
          "click", numberButtonClick)
    } else {
        // ЗАДАЧА: Это приложение было запущено заново после приостановки.
        // Восстановите здесь состояние вашего приложения
           }
           args.setPromise(WinJS.UI.processAll());
    }
};
```

В завершение этого шага добавляется всего одна строка кода, выполняемая при начальном запуске приложения. Эта строка просто регистрирует обработчик для любого события щелчка на указанной кнопке.

Рис. 1.19. Приложение для Windows 8, позволяющее получать случайные числа

Придать приложению завершающие черты можно путем повторного обращения к CSS для настройки внешнего вида надписи и кнопки. Добавьте к файлу default.css следующий код:

```
#numberButton {
   font-size: x-large;
 }
 #numberLabel {
   font-size: xx-large;
```

продолжение ⇩

```
    color: #eeee00;
    font-weight: bold;
}
```

Начальный символ решетки (#) свидетельствует о применении стиля к любому HTML-элементу, чей идентификатор соответствует указанному после решетки имени. Например, стиль, определенный как `#numberButton`, применяется ко всем элементам с идентификатором `numberButton`. Измененное приложение в действии показано на рис. 1.19.

Хотя это приложение и не отличается особой сложностью, но для начала, чтобы дать вам общее представление о подходах к программированию приложений для Windows 8 с использованием HTML5 и JavaScript, его вполне достаточно. К созданию более сложных приложений мы приступим в главе 5.

Выводы

Эта глава представляет собой пошаговое руководство по подготовке к программированию приложений для Windows 8. Сначала мы рассмотрели операционную систему и инструментарий, необходимый для написания кода, затем установили среду Visual Studio 2012 Express edition для Windows 8 и, наконец, немного повозились с простейшим приложением.

Перед тем как углубиться в разработку приложений для Windows 8, необходимо, чтобы ваш уровень знаний веб-технологий, включая HTML5, CSS и JavaScript, хотя бы минимально соответствовал всему, о чем рассказывается в этой книге. Поэтому следующие три главы дают общее представление о том, что вам нужно знать о HTML5, CSS и JavaScript, чтобы успешно работать с последующими главами. Если вы уверены, что для этого ваших знаний уже вполне достаточно, можете сразу переходить к главе 5. Если она и остальные главы впоследствии покажутся вам слишком сложными, я рекомендую еще раз просмотреть главы 2–4 и (или) освежить свои знания с помощью других информационных ресурсов, касающихся HTML5, JavaScript и CSS.

2 Знакомство с HTML5

Вообще говоря, нет ничего лучше коротких слов, а старые слова лучше всех.

Уинстон Черчилль

HTML5 является самой последней версией HTML — популярного языка, основанного на простом тексте и используемого для определения контента веб-страниц. HTML появился в начале 1990-х годов. Поначалу он был весьма простым языком разметки, подходящим для описания простых документов. Язык разметки, как таковой, основан на наборе маркеров, охватывающих текст и придающих ему особое значение.

Сначала набор элементов HTML-разметки, называемых «тегами» или (что лучше) «элементами», был крайне ограничен. Он включал в себя элементы для определения ссылок на другие документы и заголовки, добавления ссылок на изображения и абзацы и применения основных стилевых начертаний текста, таких как полужирное и курсивное. Но с годами роль HTML возросла до невообразимых высот, от простого языка описания документов до языка определения пользовательского интерфейса веб-приложений. Сегодня эта тенденция развивается с помощью HTML5.

Из самой последней версии HTML5 убраны некоторые устаревшие элементы, кроме того, стало проще хранить в одном месте элементы, предоставляющие стилевую информацию, а в другом — элементы, предоставляющие текст и определяющие его макет. В следующей главе более детально показано, что стилевая информация может быть определена через специально предназначенный для этого файл, известный как каскадная таблица стилей (Cascading Style Sheet, CSS). Кроме того, в HTML5 добавлены некоторые новые элементы, предназначенные для включения

мультимедийного контента и изображений, а также ряд новых структур для программного манипулирования контентом страницы.

Но с одним только языком HTML5 вы все же не сможете далеко продвинуться в создании полноценного приложения. А вот союз HTML5, CSS и JavaScript работает как весьма неплохое приближение к полноценному языку программирования.

- HTML5 служит для определения макета пользовательского интерфейса, а также для вставки текста и мультимедийного контента.
- CSS позволяет добавить цветовое оформление, стиль и блестящую отделку.
- И наконец, JavaScript обеспечивает добавление поведения, склеивая вместе кусочки исходных структур, таких как объектная модель документа (Document Object Model, DOM), локальное хранилище, местоположение и, к примеру, все характерные сервисы Windows 8, предоставляемые посредством библиотеки JavaScript для Windows 8 (WinJs). В частности, DOM является коллекцией программируемых объектов, раскрывающих для программистов структуру текущего документа.

На протяжении всей этой главы вкратце рассматриваются основные элементы HTML5-разметки, включая те, которые используются в формах ввода данных и в мультимедиа. Однако ни в этой, ни в других главах книги каждый аспект основ HTML не рассматривается. Если вам нужно освежить свои знания основ HTML, обратитесь к книге «Start Here! Learn HTML5», Faithe Wempen (Microsoft Press, 2012).

ВНИМАНИЕ

В этой и двух последующих главах предоставляется обзорная информация о HTML, CSS и JavaScript. В этих главах вы познакомитесь с новыми ключевыми элементами HTML5 и CSS3, получите самые необходимые знания основных технологий программирования, используемых в JavaScript. Содержимое этих глав не имеет прямого отношения к приложениям для Windows 8, а является введением в материал последующих глав, где в основной структуре HTML используются специальные элементы из библиотеки WinJs, для графического оформления применяются CSS-стили, а для задания поведения программы служит обычный JavaScript-код.

Элементы веб-страниц

Язык HTML5 появился почти на десять лет позже своего непосредственного предшественника (HTML4). Глядя на новинки HTML5, можно с полным правом сказать, что эти годы прошли не напрасно. HTML5 предоставляет набор новых элементов, предлагающих некую базовую функциональность, которую разработчики и дизайнеры могут использовать при программирования, создавая искусные и хитроумные комбинации из существующих элементов. Здесь представлен обзор всего того, что имеет отношение к созданию веб-страницы с помощью HTML5.

Создание макета страницы с помощью HTML5

В годы зарождения Интернета большинство страниц разрабатывалось в виде текстовых документов, а это означало, что их контент создавался в вертикальной ориентации в виде одного логического столбца. С годами макет страницы становился все сложнее. Сейчас чаще всего встречаются макеты с двумя и тремя столбцами, в которых зачастую имеются заголовки и колонтитулы, охватывающие логические столбцы. На рис. 2.1 показана разница между макетами, заметная с первого взгляда.

Рис. 2.1. Различные макеты HTML-страницы

Для создания столь сложных макетов с использованием основных блочных HTML-элементов, таких как `div`, разработчик должен проявить достаточную изобретательность.

ПРИМЕЧАНИЕ

В HTML блочным называют элемент, контент которого визуализируется между двумя разрывами строк — одним до, одним после контента. То есть контент выводится в виде отдельного «блока». Одним из наиболее распространенных блочных элементов является H1, который визуализирует свой текст в виде заголовка первого уровня. Другим наиболее распространенным блочным элементом является элемент div, предназначенный для создания блоков любого допустимого HTML-контента. Блочные элементы противопоставляются линейным элементам, то есть элементам, контент которых распространяется далее по странице без разрывов строк.

ВНИМАНИЕ

Следует иметь в виду, что в данной книге, как и в других книгах и статьях, такие часто встречающиеся элементы, как div и h1, зачастую упоминаются в тексте без угловых скобок. Однако включение этих элементов в HTML-разметку предполагает обязательное использование угловых скобок.

Макет, состоящий из нескольких столбцов, считается в HTML5 наиболее распространенным, поэтому он получил поддержку в виде нескольких новых специальных элементов разметки.

Подготовка учебного приложения

Примеры, с которыми вам придется работать в данной главе, являются простыми HTML-страницами, демонстрирующими некоторые функциональные возможности, доступные в HTML5 при поддержке Internet Explorer 10. Специализированное приложение для Windows 8 с целью демонстрации каждой функциональной возможности создаваться не будет, но для этого начального примера, чтобы освежить материал главы 1, мы создадим страницу-контейнер для Windows 8, связывающую вместе все ссылки на различные автономные HTML5-страницы.

Откройте Visual Studio и создайте новое пустое приложение. Назовите его Html5-Demos. После этого добавьте следующий код к телу страницы, находящейся в файле default.html, который послужит в качестве основного меню для навигации по всем HTML5-страницам, которые будут создаваться при изучении главы:

```
<body>
    <header>
      Start Here! Build <b>Windows 8</b> Applications with <b>HTML5</b> and
      <b>JavaScript</b>
      <hr />
      HTML5 samples
    </header>
    <div id="links">
      <a href="pages/multi.html">MULTI</a>
      <!—- Добавляйте сюда другие ссылки на HTML-страницы по мере их создания далее
   в этой главе. -->
    </div>
    <footer>
      <hr />
      Dino Esposito | Francesco Esposito
    </footer>
</body>
```

Получившееся приложение показано на рис. 2.2. Щелкая на ссылках (таких, как ссылка MULTI на рисунке), вы заставляете операционную систему открывать веб-страницу в Internet Explorer 10 — штатном браузере Windows 8.

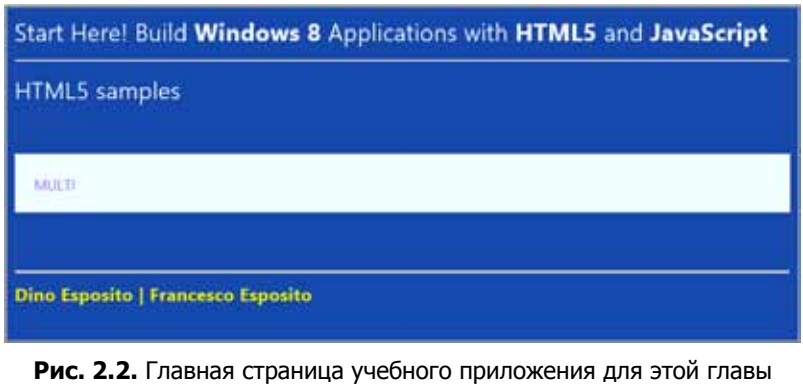

Рис. 2.2. Главная страница учебного приложения для этой главы

В дальнейшем при создании простых HTML5-страниц к телу страницы default.html будет добавляться тег-указатель <a>.

От общих блоков к семантическим элементам

В большинстве имеющихся веб-сайтов используется общепринятый макет, включающий в себя заголовок и колонтитул, а также панель навигации в левой части страницы. Чаще всего такой макет создается на основе элементов div, которым задается стиль, заставляющий их прижиматься к левому или правому краю страницы.

Давайте добавим в проект новую HTML-страницу: щелкните правой кнопкой мыши на узле проекта в проводнике решений (Solution Explorer) и выберите команду Add (Добавить) ▸ New Item (Новый элемент) в появившемся меню. Должна появиться страница, показанная на рис. 2.3. Далее в этом окне нужно выбрать пункт, позволяющий создать новую HTML-страницу, и сохранить эту страницу под именем multi.html.

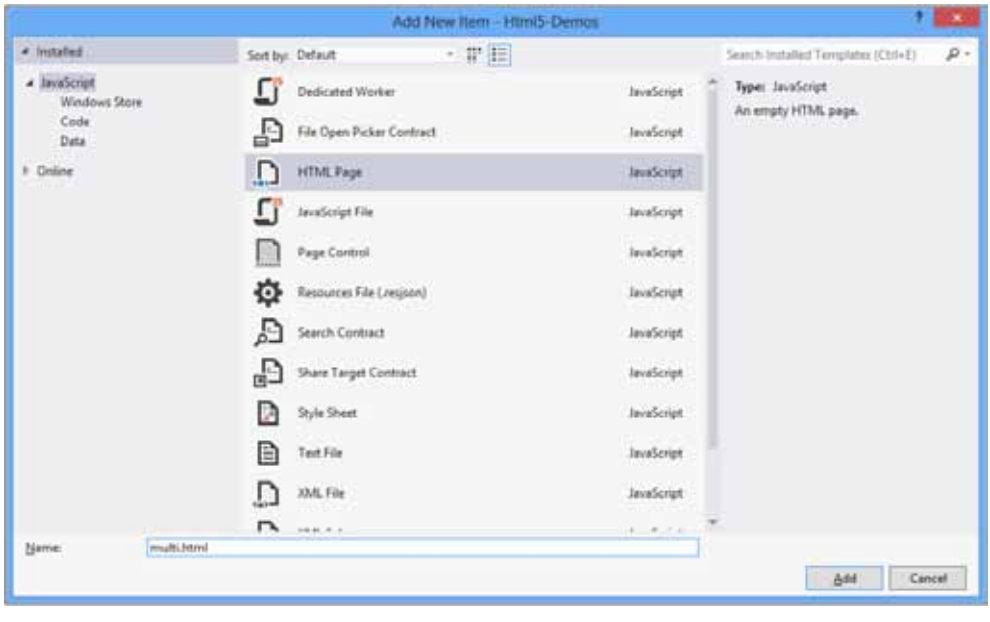

Рис. 2.3. Создание новой HTML-страницы в Visual Studio

Затем в Visual Studio нужно дважды щелкнуть на значке только что созданной HTML-страницы и заменить содержимое следующей разметкой:

```
<!DOCTYPE html>
 <html>
   <head>
     <title>MULTI-COLUMN LAYOUT</title>
   </head>
   <body>
```

продолжение ⇩

```
    <a href="/default.html">Back</a>
    <hr />
    <div id="page">
      <div id="header">
        Header of the page
        <hr />
      </div>
      <div id="navigation-bar">
        <ul>
          <li> Home </li>
          <li> Find us </li>
          <li> Job opportunities </li>
        </ul>
      </div>
      <div id="container">
        <div id="left-sidebar">
          Left sidebar
          <ul>
            <li> Article #1 </li>
            <li> Article #2 </li>
            <li> Article #3 </li>
          </ul>
        </div>
        <div id="content">
           his is the main content of the page
        </div>
        <div id="right-sidebar">
           Right sidebar
        </div>
      </div>
      <div id="footer">
        <hr />
        Footer of the page
      </div>
   </div>
 </body>
</html>
```

Атрибутам `id`, которые принадлежат элементам `div`, даны имена, объясняющие предназначение этих элементов, что помогает разобраться в намеченной для них роли. Таким образом, HTML-страница включает в себя заголовок (header), панель навигации (navigation bar), колонтитул (footer) и разметку из трех столбцов, которая находится внутри элемента с идентификатором `container`. Внешний вид страницы multi.html при ее выводе в Internet Explorer показан на рис. 2.4.

Но одна только предыдущая разметка не даст ожидаемых результатов, и страница не появится в виде макета из нескольких столбцов. Для получения нужного результата к отдельным элементам `div` следует добавить специальные графические стили, чтобы они превратились в плавающие элементы и были прижаты к левому или правому краю страницы. Графический стиль к HTML-странице добавляется с помощью CSS-разметки, помещаемой в CSS-файл. Краткий обзор CSS вы найдете в следующей главе, а сейчас у нас немного другая цель.

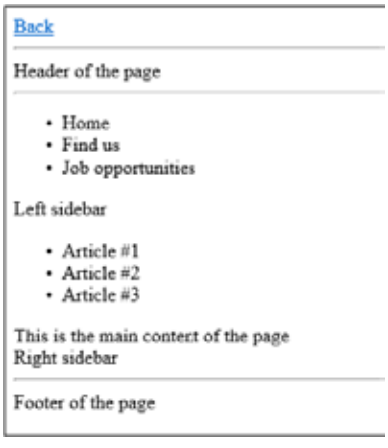

Рис. 2.4. Внешний вид страницы multi.html при ее выводе в Internet Explorer 10

Как видите, элементы `div` отличаются друг от друга по имени их атрибута `id`. Ведь каждый элемент `div` играет вполне понятную роль, которая отличает его от других элементов — заголовок отличается от колонтитула, а оба они отличаются от левой или правой боковой панели.

Элементы заголовков и колонтитулов

В HTML5 появились новые блочные элементы, имеющие конкретные имена и вполне определенное поведение. Набор новых элементов появился в результате изучения наиболее распространенных макетов, используемых авторами страниц. Например, в HTML5 появились новые вполне понятные блочные элементы `header` (для заголовка) и `footer` (для колонтитула). Подобные элементы существуют для большинства семантических элементов в предыдущем листинге. Посмотрите, как можно переписать страницу multi.html, использовав только специальные элементы, определенные в HTML5. Назовите эту страницу multi5.html. В следующем листинге показано содержимое тега `body`, принадлежащего новой странице:

```
<header>
   Header of the page
   <hr />
 </header>
 <nav>
   <a href="..."> Home </a>
   <a href="..."> Find us </a>
   <a href="..."> Job opportunities </a>
 </nav>
 <article>
   <aside>
     Left sidebar
     <ul>
       <li> Article #1 </li>
```

продолжение ⇩

```
        <li> Article #2 </li>
        <li> Article #3 </li>
      </ul>
    </aside>
    <article>
      <h1>Article #1</h1>
      <hr />
      <section> Introduction </section>
      <section> First section </section>
      <section> Second section </section>
    </article>
    <aside>
      Right sidebar
    </aside>
  </article>
  <footer>
    <hr />
    Footer of the page
</footer>
```

Заголовок и колонтитул можно вставить, использовав специальные элементы с очень простым синтаксисом:

```
<header> Markup </header>
<footer> Markup </footer>
```

Интересно заметить, что на вашей HTML5-странице может быть несколько элементов `header` и `footer`. Чаще всего они служат для создания на странице заголовка и колонтитула. Но эти элементы могут рассматриваться как блоки, предназначенные для представления заглавного контента страницы или раздела страницы и колонтитулов.

Элементы разделов и статей

В HTML5 определены два похожих элемента для представления контента страницы. Элемент `<section>` имеет более универсальный характер, поскольку предназначен для определения логических разделов HTML-страницы. Логический раздел может быть содержимым вкладки на странице, созданной в виде набора вкладок.

В то же время логический раздел может также быть частью основного контента, выводимого на странице. В таком случае элемент `section` похож на элемент, встроенный в элемент `article`:

```
<article>
    <h1>Article #1</h1>
    <hr />
    <section> Introduction </section>
    <section> First section </section>
    <section> Second section </section>
</article>
```

ПРИМЕЧАНИЕ

Элементы section, article, header и footer являются семантическими в том смысле, что браузеры считают их блочными. Если посмотреть на конечные результаты, то между семантическими элементами и обычными элементами div нет почти никакой разницы. Наиболее существенная разница заключается в выразительности получающейся разметки. При использовании элементов section, article, header и footer разметку становится проще читать, понимать и, со временем, поддерживать.

Элемент aside

На многих HTML-страницах часть контента выводится в расположенных рядом столбцах. Элемент `aside` был введен в HTML5, чтобы быстро задать контент, выводимый вокруг текущего контента. Элемент `aside` имеет весьма простой синтаксис:

```
<aside> Markup </aside>
```

Чаще всего элемент `aside` используется для определения боковой панели в элементе `article`, и, в более общем плане, для создания макетов, состоящих из нескольких столбцов контента страницы или раздела страницы.

Элемент nav

Элемент `nav` обозначает специальный раздел контента страницы — раздел, содержащий главные навигационные ссылки. Следует заметить, что не все ссылки, которые могут иметься на HTML-странице, должны быть определены внутри элемента `nav`. Этот элемент предназначен только для наиболее существенных ссылок, например тех, которые помещаются в навигационное меню главной страницы.

Синтаксис элемента `nav` интуитивно понятен. Он состоит из списка элементов-указателей `a`, находящегося внутри элемента `nav`:

```
<nav>
   <a href="..."> Home </a>
   <a href="..."> Find us </a>
   <a href="..."> Job opportunities </a>
</nav>
```

В HTML5 элемент `nav` играет весьма важную роль, так как он обозначает границы раздела страницы, в котором содержатся навигационные ссылки. Это позволяет специальным средствам чтения страницы, например браузерам для пользователей с ограниченными возможностями, лучше понимать структуру страницы и выборочно пропускать определенные части контента.

ВНИМАНИЕ

Все семантические элементы в HTML5 важно сделать более доступными и именно по этой причине нужно предусмотреть возможность их чтения пользователями с ограниченными возможностями.

Перечень других новых элементов

Семантические блочные элементы представляют собой самое большое семейство новых элементов в HTML5. Как уже отмечалось, семантические элементы важны не столько эффектом, производимым на странице, сколько более понятной разметкой страницы для разработчиков, программ и браузеров для пользователей с ограниченными возможностями.

Сами по себе семантические блочные элементы не вносят существенных изменений в то, как HTML5-совместимые браузеры визуализируют страницу. Например, для цветового оформления и позиционирования боковой панели (то есть элемента `aside`) в нужном месте следует все же прибегать к CSS. Тем не менее использование семантических элементов сокращает неразбериху, вызываемую наличием слишком большого количества универсальных блочных элементов `div`, чью роль и область действия с ходу понять непросто.

Кроме семантических элементов язык HTML5 предоставляет ряд новых элементов со встроенным поведением, которое нельзя было получить в более ранних версиях HTML, не прибегая к возможностям CSS, разметки и JavaScript. Давайте рассмотрим несколько примеров.

Элемент details

Довольно часто на странице бывают фрагменты контента, которые нужно показывать или прятать по желанию. Хорошим примером может послужить заголовок какой-то новости и весь ее контент. Иногда нужно показывать только заголовок, но при этом оставить пользователям возможность щелчком раскрыть контент, а затем спрятать его для получения доступа к дополнительному пространству.

До выхода HTML5 приходилось программировать все это вручную, используя возможности HTML, CSS и JavaScript. В HTML5 вся логика действий возложена на браузер, а программисту остается только набрать для HTML-страницы следующий код:

```
<details open>
   <summary>This is the title</summary>
   <div>
     This is the text of the news and was initially kept hidden from view
   </div>
</details>
```

Элемент `details` интерпретируется браузером и используется для реализации панели, способной исчезать и появляться. Атрибут `open` обозначает начальное состояние панели (видима либо невидима). Дочерний элемент `summary` визуализирует текст в месте, которое реагирует на щелчок, а весь остальной контент по желанию либо выводится, либо нет. Учтите, что все части элемента `details` могут в дальнейшем стилизоваться с помощью CSS.

ВНИМАНИЕ

Хотя редактор Visual Studio распознает элемент details и даже предлагает для него механизм IntelliSense, в Internet Explorer 10 этот элемент не поддерживается. Тем не менее другие HTML5-совместимые браузеры поддерживают этот элемент, в частности, это касается самых последних версий Chrome и Opera.

Элемент mark

В HTML5 добавлен также элемент `mark`, позволяющий выделять небольшие фрагменты текста по образу и подобию маркера на листе бумаги. Пользоваться им очень просто, для этого достаточно заключить какой-нибудь текст в элемент `mark`:

```
The <mark>DETAILS</mark>element is not supported by Internet Explorer 10.
```

Весь текст, за исключением той части, которая заключена в элементе `mark`, будет выведен с использованием исходных параметров. Большинство браузеров, совместимых с HTML5, предлагают для маркированного текста исходные графические параметры. Чаще всего они определены в виде желтого фона. Наверное, излишне говорить, что графические эффекты элемента `mark` могут быть изменены посредством CSS.

Вид предыдущего текста в Internet Explorer 10 показан на рис. 2.5.

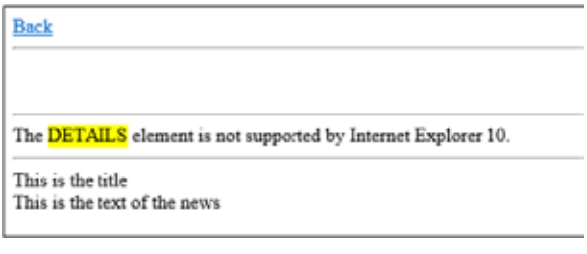

Рис. 2.5. Элемент mark в действии

Элемент datalist

С давних времен HTML-разработчики настойчиво просили предоставить им список предопределенных вариантов текстового поля. Найти случаи применения таких вариантов довольно легко. Представьте, что от пользователя требуется набрать в текстовом поле название города. С позиции автора страницы вам хочется оставить за пользователем возможность ввода любого текста, но в то же время неплохо бы оставить ему несколько предопределенных вариантов, из которых одним щелчком можно выбрать нужный. До выпуска HTML5 это требовало программирования на JavaScript, поскольку изначально язык HTML предоставлял только два варианта:

свободный текст без функции автозавершения и фиксированный список вариантов, не допускающий свободного набора текста. Этот пробел заполняет новый элемент `datalist`. Скопируйте следующий текст в тело новой HTML-страницы по имени datalist.html:

```
<input list="cities" />
 <datalist id="cities">
   <option value="Rome">
   <option value="New York">
   <option value="London">
   <option value="Paris">
</datalist>
```

В этом примере элемент `datalist` привязан к конкретному полю ввода под названием `cities`. Интересно заметить, что привязка осуществляется с помощью нового атрибута, определяемого в элементе `input`: этот атрибут называется `list`. Атрибут получает имя элемента `datalist`, который будет использоваться в качестве источника вариантов ввода.

Когда поле ввода получает фокус, содержимое элемента `datalist` служит для автозавершения ввода пользователя. На рис. 2.6 этот элемент показан в действии в браузере Internet Explorer.

Рис. 2.6. Элемент datalist в действии

Элементы прежних версий, отсутствующие в HTML5

В язык HTML5 добавлен ряд новых элементов, но из него также удален и ряд старых элементов, чье присутствие в сочетании с новыми CSS-средствами и HTML5-элементами привело бы к функциональной избыточности.

В список наиболее заметных более не поддерживаемых элементов включены элементы `frame` и `font`. Однако следует заметить, что элемент `iframe` по-прежнему доступен.

Кроме того, удален ряд элементов, имеющих отношение к стилевому оформлению, среди них элементы `center`, `u` и `big`. Дело в том, что их функциональные возможности могут быть легко получены с помощью CSS. Возможно, из-за более широкой, с годами сложившейся практики использования авторами страниц, в HTML5 оставлена поддержка таких элементов как `b` (для полужирного начертания) и `i` (для курсива), которые являются логическими эквивалентами более неподдерживаемых элементов `u` и `big`.

Сбор данных

Изначально HTML был создан в качестве языка для создания гипертекстовых документов. С годами язык был обогащен возможностями разметки и основными средствами сбора данных. Написание форм ввода для сбора данных от пользователей оказалось непростой задачей. Одно дело собирать обычный текст и совершенно другое — осуществлять сбор дат, чисел или адресов электронной почты.

На протяжении довольно длительного периода времени в HTML предлагались лишь текстовые поля ввода, в которых совершенно невозможно было отличить от обычного текста числа, даты и адреса электронной почты. Впоследствии разработчикам страниц приходилось придумывать механизмы предотвращения ввода пользователями ненадлежащих символов и проверки допустимости введенного текста на стороне клиента.

С выпуском HTML5 основная часть данной работы была возложена на браузер. Это означает, что простое использование несколько более сложного набора элементов позволяет разработчикам получить такой же уровень проверки формы, но более быстрым и безопасным образом.

Адаптация полей ввода

В HTML5 формы ввода данных по-прежнему создаются с помощью элементов разметки, к которым вы привыкли в предыдущих версиях языка. То есть следующая разметка, как и прежде, даст вам возможность выкладывать данные на заданный сервер и набирать для него контент:

```
<form action="http://www.yourserver.com/upload">
   <span> Your name </span>
   <input type="text" value="" />
   <input type="submit" value="Save" />
</form>
```

Элемент `input` вставляет графический элемент (например, поле ввода или раскрывающийся список) для сбора вводимых данных. Элемент `input` используется также для добавления кнопки, инициирующей процесс отправки данных на сервер.

В HTML5 элемент `input` поставляется с расширенными возможностями для полей ввода. Например, браузер может предложить вам средство выбора даты, ползунки и поля поиска. В то же время, браузер предлагает независимую проверку формы для большинства возможных ситуаций, например, когда поле является обязательным и не может быть оставлено пользователем незаполненным.

Новые типы ввода

Если посмотреть на синтаксис элемента `input` в HTML5, то основное отличие от предыдущих версий заключается в списке тех значений, которые теперь допустимы для атрибута `type`. Некоторые новые типы ввода, поддерживаемые в HTML5, перечислены в табл. 2.1.

Таблица 2.1. Значения атрибута type, доступные в HTML5

Значение	Описание
Color	Браузер может визуализировать любой элемент пользовательского интерфейса, который позволяет задать цвет. Примечание: этот тип ввода в Internet Explorer 10 не поддерживается
date	Браузер может визуализировать любой элемент пользовательского интерфейса, который позволяет задать дату
email	Браузер может визуализировать любой элемент пользовательского интерфейса, который позволяет задать адрес электронной почты
number	Браузер может визуализировать любой элемент пользовательского интерфейса, позволяющий вводить числа
range	Браузер может визуализировать любой элемент пользовательского интерфейса, позволяющий вводить диапазон чисел
search	Браузер может визуализировать любой элемент пользовательского интерфейса, позволяющий вводить искомый текст
tel	Браузер может визуализировать любой элемент пользовательского интерфейса, позволяющий вводить телефонный номер
time	Браузер может визуализировать любой элемент пользовательского интерфейса, позволяющий вводить время
url	Браузер может визуализировать любой элемент пользовательского интерфейса, позволяющий вводить URL-адрес

Учтите, что список, представленный в табл. 2.1, не полон, в нем перечислены только те типы вводимых данных, поддержку которых сегодня действительно обеспечивают некоторые веб-браузеры. Другие типы вводимых данных (например, `week` для ввода дня недели), хотя и являются частью текущего проекта HTML5, нигде не реализованы. Подробности можно найти по адресу http://www.w3schools.com/html5.

НАСКОЛЬКО БРАУЗЕРЫ ОПРАВДЫВАЮТ ВОЗЛАГАЕМЫЕ НА НИХ ОЖИДАНИЯ

Представленные в табл. 2.1 сведения соответствуют стандарту HTML5, следование которому ожидалось на момент его официального утверждения и обнародования. Похоже, что в данное время наилучшую поддержку текущего проекта HTML5 предоставляют браузеры Opera и Chrome. В целом, нынешняя поддержка HTML5 от всех браузеров не является единообразной.

Дело в том, что HTML5 не является согласованным стандартом и не будет таковым еще в течение нескольких лет. В то же время, компании стремятся использовать все лучшее, что есть в HTML5 в действующих веб-сайтах и приложениях. Но этим веб-сайтам и приложениям требуются совместимые браузеры. Представляется, что в создавшейся ситуации производители браузеров играют в своеобразные догонялки. С каждым новым выпуском того или иного браузера улучшается поддержка стандарта HTML5 или приспособленность к текущему стандарту с отказом от возможных свойств, появившихся в результате неправильной интерпретации предыдущих состояний проекта.

Однозначного определения HTML5 нельзя будет получить в течение еще нескольких лет. Internet Explorer 10, являющийся браузером, интегрированным в Windows 8, имеет по сравнению с Internet Explorer 9 существенно улучшенную поддержку HTML5. Следует заметить, что в Windows 8 можно найти две разновидности браузера Internet Explorer: классическую версию, являющуюся надстройкой над Internet Explorer 9, и версию, имеющую сценарий диалога с пользователем, присущий Windows 8. В этой версии, в частности, отсутствует возможность запуска дополнительных модулей (таких, как Flash и Silverlight) и организации избранного в папках.

Придание полям ввода свойства автоматического получения фокуса

В HTML5 предлагается вполне определенное решение двух весьма распространенных проблем, с которыми разработчики сталкивались на протяжении многих лет и которые они решали с помощью JavaScript-кода. Первая проблема касается передаче полю фокуса ввода.

Используя JavaScript, можно заставить браузер установить фокус ввода на конкретное поле ввода из тех, которые выводятся на странице. В HTML5 для этого можно воспользоваться новым атрибутом элемента `input`, который называется `autofocus`. Попробуйте поместить в тело новой HTML-страницы под названием autofocus.html следующий код:

```
<form>
   <input type="text" value="Dino" />
   <input type="text" autofocus />
```

продолжение ⇗

```
    <br />
    <input type="submit" value="Save" />
</form>
```

Сохраните страницу и выведите ее в Internet Explorer. Как показано на рис. 2.7, курсор свидетельствует о том, что фокус ввода находится во втором поле.

Рис. 2.7. Атрибут autofocus в действии

Предоставление пользователям подсказок

При просмотре рис. 2.7 довольно трудно определить, какой именно контент должен вводиться в то или иное поле. Наверное, на настоящей странице, чтобы пользователям было понятнее, какой контент в каком поле ожидается, должны быть надписи и более сложная разметка. Это и есть та самая вторая проблема, о которой я упоминал совсем недавно.

В последнее время разработчики взяли в привычку выводить короткие текстовые сообщения в поле ввода текста, давая пользователям своеобразную инструкцию. Перед выпуском HTML5 такую функциональность можно было реализовать только с помощью JavaScript-кода. В HTML5 благодаря атрибуту `placeholder` все делается намного проще и естественнее.

Создайте новую HTML-страницу и сохраните ее в файле placeholder.html. Теперь отредактируйте разметку страницы следующим образом:

```
<form>
    <input type="text" placeholder="First name" />
    <input type="text" placeholder="Last name" />
    <br />
    <input type="submit" value="Save" />
</form>
```

Как показано на рис. 2.8, теперь подсказка об ожидаемом контенте есть в обоих пустых полях.

Рис. 2.8. Атрибут placeholder в действии

Отправка данных формы

Иногда разработчики не имеют никакой другой возможности, кроме как, сдерживая раздражение, многократно писать один и тот же эталонный код. Хорошим примером такого эталонного кода, от которого давно пора бы избавиться, может послужить проверка форм ввода на HTML-страницах. Любые данные, собранные в HTML-формах ввода, прежде чем они будут задействованы в некоторых коммерческих задачах, подлежат тщательной проверке на сервере. Однако некоторые основные задачи по проверке данных легко могут быть переданы браузерам и обрабатываться разработчиками с помощью разметки, а не JavaScript-кода.

HTML5 помогает сократить количество эталонного кода, требуемого для создания эффективных форм ввода данных. О самых новых HTML5-атрибутах элемента `input` здесь уже упоминалось, следующим шагом будет рассмотрение других атрибутов, которыми можно воспользоваться для управления всем процессом отправки формы, включая обеспечение недопустимости отправки пустых обязательных полей и соответствие пользовательского ввода ожидаемым образцам. Например, если запрашивается телефонный номер, пользователю не должно быть разрешено вводить что-нибудь, что не является правильным телефонным номером.

Определение обязательных полей

Добавляя к элементу `input` атрибут `required`, вы сообщаете браузеру о том, что поле ввода не должно оставаться пустым, когда данные формы, содержащей это поле ввода, отправляются на сервер. Атрибут `required` используется только в том случае, если поле считается обязательным.

Рассмотрим следующую разметку HTML-страницы по имени required.html:

```
<body>
 <form>
   <input type="text" placeholder="Your PIN" required />
   <br />
```

продолжение ⇩

```
    <input type="submit" value="Enter" />
  </form>
</body>
```

Когда пользователь щелкает по кнопке отправки при пустом текстовом поле, браузер автоматически отклоняет отправку и выводит сообщение об ошибке. Конкретный контент, форма и графическое оформление сообщения об ошибке от браузера к браузеру могут меняться, но всем совместимым с HTML5 браузерам присуще одинаковое поведение. На рис. 2.9 показана реакция на пустое обязательное поле браузера Internet Explorer 10.

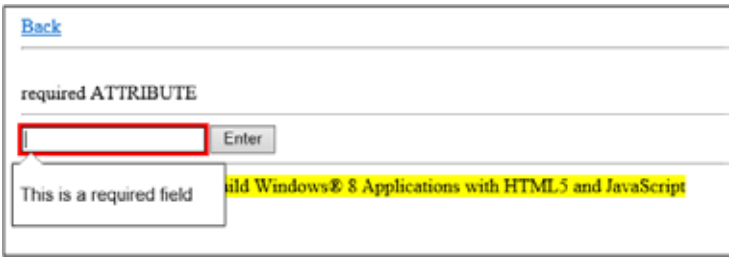

Рис. 2.9. Атрибут required в действии

Совместимые с HTML5 браузеры допускают настройку сообщения об ошибке, проводимую с помощью атрибута `oninvalid`:

```
<form>
   <input type="text" placeholder="Your PIN" required
      oninvalid="this.setCustomValidity('PIN is mandatory')" />
   <br />
   <input type="submit" value="Enter" />
</form>
```

ПРИМЕЧАНИЕ

Вообще-то, атрибут oninvalid служит для задания любого JavaScript-кода, который должен быть выполнен при недопустимом содержимом поля ввода: либо когда обязательное поле ввода оставлено пустым, либо когда его содержимое не прошло проверку на допустимость.

Проверка допустимости с использованием регулярных выражений

В табл. 2.1 были перечислены широко используемые новые типы полей ввода, поддерживаемые браузерами, совместимыми с HTML5. Если страница предназначена для сбора данных, то можно задействовать поле ввода этих данных, то есть можно использовать поле ввода числовых данных, если нужно собрать именно числовые

данные, и т. д. А что делать, если вы намереваетесь собрать данные, отформатированные каким-то особым образом, не соответствующим предопределенным типам вводимых данных? Например, что, если вам нужно, чтобы пользователи вводили строку из двух букв, за которыми строго следуют шесть цифр?

Как показано в следующем примере, в HTML5 в таком случае можно воспользоваться атрибутом `pattern`:

```
<form>
   <input type="text"
          placeholder="Your PIN"
          title="2 letters + 6 digits"
          pattern="[a-zA-Z]{2}\d{6}" />
   <br />
   <input type="submit" value="Enter" />
</form>
```

При использовании атрибута `pattern` Internet Explorer 10 требует, чтобы вы также задали атрибут `title`, который обычно используется для добавления подсказки в большинстве HTML-элементов. Текст атрибута `title` объединяется с исходным статическим сообщением для создания ответной реакции в адрес пользователя при вводе в поле неверного контента.

На рис. 2.10 показано, как Internet Explorer 10 работает с атрибутом `pattern` при попытке отправки неверного контента.

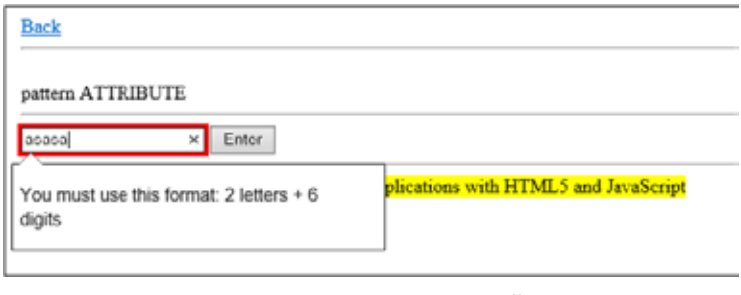

Рис. 2.10. Атрибут pattern в действии

Значением атрибута `pattern` должно быть регулярное выражение. Регулярные выражения могут быть очень сложными, а сама тема регулярных выражений достойна отдельной книги, но освоение базовых знаний в области регулярных выражений не составляет особого труда. Дополнительные сведения о регулярных выражениях можно найти по адресу http://www.regular-expressions.info.

Формы и проверка допустимости введенных данных

Каждая HTML-форма должна иметь кнопку отправки, при щелчке на которой браузер собирает содержимое полей ввода и готовит его к отправке на указанный

URL-адрес. До выпуска HTML5 браузер не отвечал за проверку содержимого формы, но разработчики могли привязать проверку к процессу отправки с помощью небольшого фрагмента JavaScript-кода.

Проверка формы заключается в проверке каждого поля ввода формы на допустимость его содержимого. Хотя стандарт HTML5 не заставляет браузеры проверять содержимое формы, именно это большинство браузеров и делает по умолчанию. Браузеры, совместимые с HTML5, позволяют отключить режим проверки всей формы, но не отдельных ее полей. Отключить режим проверки данных формы можно с помощью атрибута `novalidate`, как показано в файле novalidate.htm:

```
<form novalidate>
   <input type="text" placeholder="Your PIN"
          title="2 letters + 6 digits"
          pattern="[a-zA-Z]{2}\d{6}" />
   <br />
   <input type="submit" value="Enter" />
</form>
```

В этом случае данные формы отправляются на сервер независимо от состояния данных в полях ввода.

Если форма содержит несколько кнопок отправки, то режим проверки допустимости данных для той или иной кнопки можно включать или отключать. Чтобы отключить режим проверки данных при отправке формы при щелчке на конкретной кнопке, к ней добавляется атрибут `formnovalidate`:

```
<input type="submit" value="..." formnovalidate />
```

ПРИМЕЧАНИЕ

Атрибут formnovalidate отменяет атрибут формы novalidate, если установлены оба этих атрибута.

Мультимедийные элементы

В HTML5 предлагаются два новых элемента разметки, которые могут использоваться разработчиками для воспроизведения аудио- и видеофайлов на веб-страницах без обращения к внешним дополнительным модулям, таким как Flash и Silverlight. Вся инфраструктура, необходимая для воспроизведения аудио и видео (включая графическую обратную связь с пользователем) предлагается непосредственно браузером.

Элемент audio

Для включения аудиоконтента в HTML-документы используется элемент `audio`. Его синтаксис довольно прост:

```
<audio src="/hello.mp3">
   <p>Your browser does not support the audio element.</p>
</audio>
```

При желании в тело элемента `<audio>` может быть включена разметка для использования в случае, когда браузер не в состоянии успешно справиться с элементом `audio`. Далее вопрос вставки аудиоконтента в HTML5-страницы рассматривается более подробно.

Использование элемента audio

Элемент `audio` поддерживает множество атрибутов, перечисленных в табл. 2.2. Наиболее важным из них является атрибут `src`, предназначенный для задания места, в котором находится аудиопоток.

Таблица 2.2. Атрибуты элемента audio

Атрибут	Описание
autoplay	Показывает, что аудиоконтент начнет воспроизводиться сразу же, как только контент страницы станет доступен браузеру
controls	Заставляет браузер показать средства управления аудиоконтентом, например кнопки воспроизведения и паузы
loop	Показывает, что аудиоконтент будет автоматически перезапущен после завершения воспроизведения
preload	Сообщает браузеру о порядке загрузки аудиоконтента при загрузке страницы. Допустимыми значениями являются none (отсутствие предварительной загрузки контента), auto (при загрузке страницы должен быть загружен весь контент) и metadata (при загрузке страницы должны быть загружены только метаданные контента). Отметьте, что атрибут preload игнорируется при наличии атрибута autoplay
src	Показывает URL-адрес аудиофайла, причем как локального, так и удаленного

До сих пор я ссылался на аудиоконтент в несколько общем плане, не упоминая о конкретном аудиоформате, например MP3 или WAV. Основная проблема совместимых с HTML5 браузеров заключается в том, что не все из них поддерживают один и тот же набор аудиоформатов, предлагаемый по умолчанию (без обращения к внешним компонентам).

Проблемы кодеков

Аудиофайл является последовательностью байтов, декодируемой кодеком с целью воспроизведения. При этом аудиофайл может быть закодирован во множестве форматов, например MP3, WAV, OGG и т. д., причем для каждого требуется соответствующий кодек. Зачастую кодек является частью программного обеспечения, реализующего запатентованные алгоритмы, поэтому непосредственной вставке кодека в браузер могут препятствовать проблемы авторских прав.

Текущий стандарт HTML5 не задает официальные правила относительно кодеков, поэтому принятие решения о поддерживаемом формате возлагается на производителей браузеров.

С точки зрения разработчика, в этом нет ничего удивительного. Различные браузеры поддерживают различные аудиоформаты, то есть решение проблемы наиболее эффективного способа воспроизведения аудиоконтента одной и той же страницы на различных браузерах возлагается на разработчиков.

Поддерживаемые кодеки

Простейший подход к проблеме нескольких кодеков состоит в предоставлении нескольких файлов, чтобы браузер мог выбрать наиболее подходящий под его возможности. То есть вместо привязки элемента `audio` только к одному файлу и кодеку он привязывается сразу к нескольким источникам. При этом атрибут `src` больше не используется, а вместо него применяется набор элементов `source`, указываемых внутри элемента `audio`. Рассмотрим пример воспроизведения аудиофайла с помощью элементов `source`:

```
<audio controls autoplay>
   <source src="hello.ogg" type="audio/ogg" />
   <source src="hello.mp3" type="audio/mp3" />
   <p>Your browser does not support the audio element.</p>
</audio>
```

Элементы `<source>` привязываются к различным аудиофайлам. Для поддержки браузер задействует первый же известный ему формат. При всей простоте реализации этот подход не лишен недостатков в том смысле, что вам нужно конвертировать каждый доступный аудиофайл в несколько форматов и хранить его на сервере в нескольких копиях.

Основным ориентиром должен служить формат OGG, не отягощенный никакими патентами на программное обеспечение. Этот формат будет работать в Firefox, Opera и Chrome. А для Safari и Internet Explorer вместо него придется использовать формат MP3.

Элемент video

Для вставки видеоконтента в HTML-документ используется элемент `video`. Синтаксис этого элемента довольно прост, напоминает синтаксис элемента `audio`:

```
<video src="/hello.mp4">
   <p>Your browser does not support the video element.</p>
</video>
```

Точно так же в тело элемента `video` может быть включена разметка для использования в случае, когда браузер не в состоянии успешно справиться с элементом `video`.

Использование элемента video

В табл. 2.3 представлены атрибуты, служащие для настройки внешнего вида и поведения элемента `video` в браузерах, совместимых с HTML5.

Таблица 2.3. Атрибуты элемента video

Атрибут	Описание
autoplay	Показывает, что видеоконтент начнет воспроизводиться сразу же, как только контент страницы станет доступен браузеру
controls	Заставляет браузер показать средства управления видеоконтентом, например кнопки воспроизведения и паузы
height	Показывает желаемую высоту видеопроигрывателя в HTML-документе
loop	Показывает, что видеоконтент будет автоматически перезапущен после завершения воспроизведения
muted	Показывает, что звук в видеоклипе должен быть отключен
poster	Заставляет браузер при загрузке видеоконтента (или до тех пор, пока пользователь не включит режим воспроизведения) выводить указанное изображение
preload	Сообщает браузеру о порядке загрузки видеоконтента при загрузке страницы. Допустимыми значениями являются none (без предварительной загрузки контента), auto (при загрузке страницы должен быть загружен весь контент) и metadata (при загрузке страницы должны быть загружены только метаданные контента). Отметьте, что при наличии атрибута autoplay атрибут preload игнорируется
src	Показывает URL-адрес видеофайла, причем как локального, так и удаленного
width	Показывает желаемую ширину видеопроигрывателя в HTML-документе

Настоятельно рекомендуется всегда устанавливать для элемента `video` значения ширины и высоты с помощью аргументов `width` и `height`. Это поможет браузерам зарезервировать достаточно пространства при выводе страницы на экран. Кроме того, ширину и высоту всегда нужно устанавливать равной реальному размеру видеоклипа, планирующегося к размещению. Если уменьшить размеры видеопроигрывателя, это заставит браузер делать дополнительную работу. Имейте в виду, что уменьшение размеров видео не даст пользователю никакой экономии времени загрузки. Если видео слишком большое для страницы, то сначала его нужно уменьшить с помощью специальной программы, а затем сослаться на него, указав его новый размер.

На рис. 2.11 показано, как элемент `video` визуализируется браузером Internet Explorer 10.

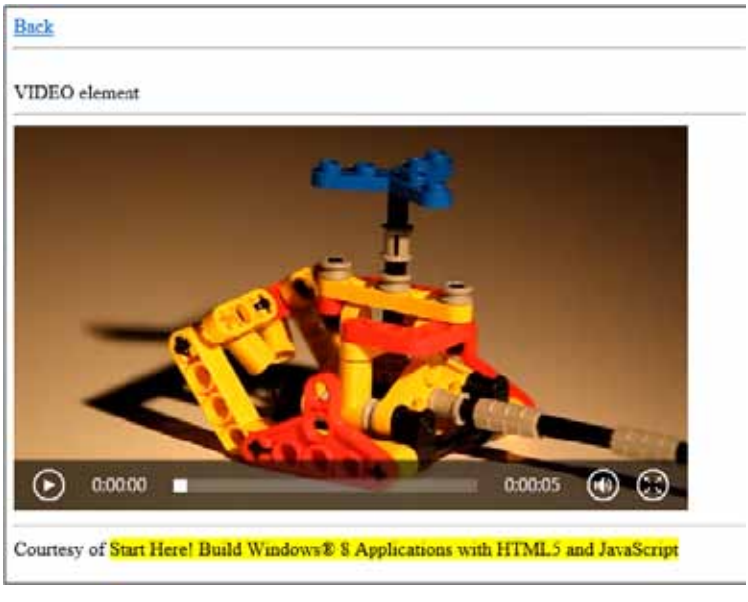

Рис. 2.11. Элемент video в действии

Поддерживаемые кодеки

Что касается кодеков, с элементом `video` связаны те же проблемы, что и с элементом `audio`. Следовательно, он требует таких же путей решения проблем.

Атрибут `src` не нужно использовать до тех пор, пока не станет известно о невозможности воспроизведения видеоконтента на некоторых браузерах. Для наиболее широкой поддержки со стороны совместимых с HTML5 браузеров нужно использовать элемент `source`. Переделанная разметка учебного файла video.html имеет следующий вид:

```
<video controls width="320" height="240">
   <source src="/sample.ogg" type="video/ogg" />
   <source src="/sample.mp4" type="video/mp4" />
   <p>Your browser does not support the video element.</p>
</video>
```

Как и для `audio`, элемент `source` является ссылкой на различные видеоклипы, и браузер выбирает первый же поддерживаемый им формат. В качестве рекомендации для Internet Explorer и Safari нужно планировать использование видеконтента в формате MP4, а для всех остальных браузеров — в формате OGG.

Выводы

HTML существует уже около двух десятилетий, но только сейчас произошли существенные изменения синтаксиса этого языка. Новая спецификация HTML5 избавлена от ряда устаревших элементов и дополнена новыми элементами разметки для решения специфических (и широко распространенных) задач. У новых элементов имеется более четкая семантика, которая проясняет их предназначение, например заголовок, колонтитул, меню, раздел и т. д.

Но эти новые элементы сосуществуют с прежними, с семантической точки зрения более универсальными элементами, такими как `div`. В результате разработчик иногда имеет несколько способов получить один и тот же вариант визуализации данных — с помощью непосредственно HTML5-элементов или сочетания более универсальных элементов. Если планируется ориентироваться на браузеры, совместимые с HTML5, то использование новых элементов упростит чтение и понимание разметки, одним словом, она станет проще.

Назначение данной книги — помощь в создании приложений для Windows 8, а не классических веб-сайтов, что нивелирует различия между HTML и HTML5: в этой книге повсеместно используется HTML5. Но если вы нацелились на создание веб-сайта, предназначенного для широкой публики, то интеграция HTML5 в разметку страниц становится значительно сложнее. Для создания веб-приложений придется учитывать различия браузеров и обеспечивать универсальность поведения сайта во всех основных браузерах.

3 Знакомство с CSS

В вопросах стиля плыви по течению;
в вопросах принципа стой твердо, как скала.

Томас Джефферсон

Из главы 2 вы узнали, что HTML-страница состоит из ряда элементов, которые в совокупности определяют контент и разметку того, что выводится браузером на экране. Каждый элемент языка HTML имеет собственные семантику и синтаксис. Поэтому элемент `input`, к примеру, обозначает поле ввода, а дополнительные атрибуты определяют форму и поведение этого поля ввода. Вид подобных элементов обычно определяется конкретным браузером.

Но до сих пор ничего еще не было сказано о том, как этим элементам придать индивидуальность. А ведь изменение шрифта, цветовых решений, отступов и размеров HTML-элементов не только возможно, причем в довольно большой степени, но и желательно. Акроним CSS (Cascading Style Sheet — каскадная таблица стилей) является названием языка, используемого для форматирования контента HTML-страниц.

Вид HTML-страницы определяют три компонента: контент, стиль и поведение. Контент, как отмечалось в главе 2, визуализируется с помощью языка HTML. Стиль, как будет показано в данной главе, задается с помощью CSS. И наконец, поведением управляет язык JavaScript, речь о котором пойдет в следующей главе.

Стилизация веб-страниц

Язык HTML появился задолго до CSS, в начале 1990-х годов. Сначала разработчики для задания внешнего вида страницы просто добавляли к элементам разметки специальные атрибуты. Будучи вполне эффективным на первых порах, такой подход

вскоре стал неуправляемым и превратился в источник путаницы для разработчиков и пользователей.

Для захвата все больших долей рынка производители с каждым новым выпуском своих браузеров стали добавлять новые собственные стилевые атрибуты. В качестве побочного, хотя и не второстепенного, эффекта HTML-документы стали значительно больше по размеру, что повлекло за собой проблемы при их загрузке. Работы серверам прибавилось, но ее также прибавилось и клиентским браузерам, что замедлило поступление ответов пользователям.

Кроме того, браузерам часто приходилось интерпретировать неизвестные атрибуты и теги. Разработчикам не просто было решить, какой должна быть реакция на непонятную разметку: выдача пользователю сообщения об ошибке или просто пропуск неизвестного элемента. Для большинства браузеров был выбран второй вариант. Хотя этот выбор не портил пользователям впечатление от просмотра сайта, он существенно усложнял жизнь разработчикам, и, если оглянуться назад, не будет преувеличением сказать, что он позволил на несколько лет отсрочить возникновение взрывоопасной ситуации в веб-разработке на стороне клиента.

В поиске согласованного способа разделения контента и его представления в середине 1990-х годов консорциум World Wide Web (W3C) организовал комитет по созданию некоего стандартного языка для стилизации контента веб-страниц. Так родился язык CSS. Самые первые рекомендации были выпущены в 1996 году. В последующие годы постоянно появлялись новые уровни спецификаций наряду с ростом их поддержки со стороны производителей браузеров и все возрастающей степени их использования разработчиками. В настоящее время уже завершена работа над выработкой и широкой поддержкой браузерами стандарта CSS3. Идет работа и над стандартом CSS4.

В данной книге все обозначаемое как CSS относится к CSS3.

Добавление к страницам CSS-информации

Итак, CSS является языком, дополняющим HTML и открывающим перед создателями страниц широкие возможности по определению структуры и контента в одном файле, а разметки и внешнего вида — в другом. Благодаря столь тонкому разделению зон ответственности, разные команды специалистов, обычно веб-разработчиков и дизайнеров, могут работать параллельно над одним и тем же проектом.

В данной главе исследуются различные способы добавления CSS-стилей к HTML-страницам.

Встраиваемая стилизация

CSS позволяет добавлять стилевую информацию к отдельным HTML-элементам. Вплоть до нескольких последних лет среди авторов HTML-страниц было вполне

обычным делом вставлять стилевую информацию внутрь определения HTML-элемента, то есть локально. При таком подходе, известном как *встраиваемая стилизация* (inline styling), к каждому настраиваемому HTML-элементу добавляется новый атрибут `style`. Рассмотрим следующий пример:

```
<div style=" ... ">
   <!-- сюда помещается какая-нибудь разметка -->
</div>
```

Содержимым атрибута `style` является строка с точкой с запятой (`;`) в качестве разделителя. Каждая лексема внутри такой строки состоит из двух частей: допустимого имени и значения, отделенных друг от друга двоеточием (`:`). Типовая строка стиля, определяющая цвета первого плана и фона любого контента внутри стилизуемого элемента `div`, имеет следующий вид:

```
<div style="background-color:#000000;color:#ffffff">
   <!-- сюда помещается какая-нибудь разметка -->
</div>
```

Конечным результатом выполнения предыдущего фрагмента кода является придание фону контента элемента `div` черного цвета (`#000000`) наряду с приданием белого цвета (`#ffffff`) всему тексту, содержащемуся в этом элементе. Эта стилевая информация касается конкретного элемента `div` и никак не сказывается на контенте страницы, находящемся за его пределами.

У этой технологии есть свои положительные и отрицательные черты, но отрицательные перевешивают. И тем не менее встраиваемая стилизация все еще применяется, она вполне понятна и ее быстро осваивают как опытные разработчики, так и новички. Но у нее есть весьма существенные недостатки. В частности, объем разметки страниц быстро растет, и вскоре ее становится трудно читать и понимать. Кроме того, у нее имеется существенный потенциал для повторений стилевых определений как на одной и той же, так и на нескольких страницах. Короче, встраиваемая стилизация считается в программировании дурным тоном и ее следует избегать или же применять только в самом крайнем случае в ограниченном количестве мест, в которых нужно сделать исключение из более общих стилевых правил, задаваемых средствами CSS.

Внедренные стили

Следующий вариант заключается в группировке стилевых определений в нескольких местах внутри HTML-файла. В качестве хранилища специальных стилей можно использовать элемент `style`. Внутри элемента `style` сначала идентифицируется целевой элемент, а затем определяются атрибуты, влияющие на внешний вид целевого элемента. Элемент `style` обычно следует за элементом `head` в верхней части HTML-страницы. На любой веб-странице может быть несколько элементов `style`. Рассмотрим пример:

```
<html>
   <head>
     <style>
       body {
         background-color: black;
         color: white;
       }
     </style>
   </head>
 ...
</html>
```

Нетрудно понять, что текст внутри фигурных скобок описывает применяемый стиль. Выражение, которое находится непосредственно перед открывающей скобкой, в предыдущем примере это тег `body`, указывает на элемент (или элементы), к которому должен применяться стиль.

Совсем скоро этот вопрос будет рассмотрен более подробно, а сейчас важно понять, что выражение, идентифицирующее цель стилизации, называется *селектором* (selector) и может быть именем определенного пользователем CSS-класса, идентификатором конкретного элемента или названием тега HTML-элемента. В предыдущем примере `body` является названием основного HTML-элемента; следовательно, цвета фона и переднего плана, установленные с помощью этого стиля, распространяются на все тело страницы.

Хотя элементы `style`, внедряемые в HTML-страницу, обеспечивают более четкое, чем при встраиваемой стилизации, выражение авторского замысла, их использование не приветствуется. И хотя по сравнению со встраиваемой стилизацией внедряемые стили способствуют многократному использованию стилей на одной и той же странице, стилевую информацию приходится внедрять в каждую страницу, что делает страницы более громоздкими.

То есть многократное использование стилей возможно, но только в пределах отдельной страницы, а для задания аналогичного стиля на разных страницах приходится повторять определение стиля на каждой такой странице.

Использование внешних файлов

Вместо внедрения стилей или встраивания тегов `style` при стилизации HTML-элементов рекомендуется использовать отдельные CSS-файлы, или *таблицы стилей* (style sheets), с автономными ссылками на них на каждой странице. Браузер идентифицирует CSS-контент в виде отдельного URL-адреса и загружает его только один раз независимо от того, сколько страниц на веб-сайте используют стили из загруженной таблицы стилей. Более того, загруженный файл может быть кэширован на локальном компьютере и использоваться снова и снова, пока не истечет срок его хранения в кэше, то есть издержки на загрузку стилей браузерем исключаются.

Рассмотрим небольшой пример определения и ссылки на таблицу стилей для изменения исходного вида HTML-страницы. Сначала нужно в диалоговом окне, показанном на рис. 3.1, выбрать режим создания новой HTML-страницы с именем demo1.html, а затем перейти к созданию новой таблицы стилей (вариант Style Sheet) с именем demo1.css.

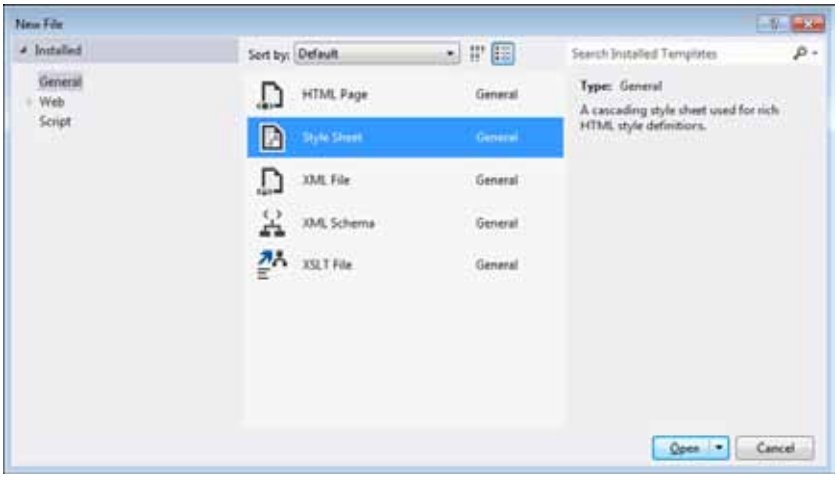

Рис. 3.1. Создание новой HTML-страницы и таблицы стилей

Теперь откройте HTML-страницу и отредактируйте ее контент, придав ему следующий вид:

```
<!DOCTYPE html>
 <html>
   <head>
     <title>CSS Example</title>
   </head>
   <body>
     <header>This is our header</header>
     <footer>
       <hr />
       Start Here! Build Windows® 8 Applications with HTML5 and JavaScript
     </footer>
   </body>
</html>
```

Пока страница всего лишь наполнена скромным контентом, а для связи HTML-страницы с какой-либо таблицей стилей еще ничего не сделано. Чтобы исправить ситуацию, нужно следующим образом отредактировать элемент head этой страницы:

```
<head>
     <title>CSS Example</title>
     <link href="demo1.css" rel="stylesheet" />
</head>
```

Теперь при выводе страницы на экран браузер для настройки внешнего вида элементов воспользуется информацией, содержащейся в таблице стилей.

ПРИМЕЧАНИЕ

Далее в этой главе, как и во всей книге, будут всегда использоваться таблицы стилей, находящиеся в отдельных CSS-файлах. Если не указано что-либо иное, встраиваемая стилизация или внедренные стили больше применяться не будут.

Выбор стилизуемых элементов

Следующим этапом обучения является создание таблицы стилей. Эта таблица состоит из последовательности команд, определяемых в соответствии со следующим образцом:

```
селектор {свойство1: значение1; свойство2: значение2; ...}
```

Селектор (selector) — это выражение, идентифицирующее стилизуемый элемент (или элементы). Селектор может идентифицировать отдельный элемент по его уникальному идентификатору, все элементы указанного тега (например, все элементы `div`) или все элементы, использующие один и тот же CSS-класс. Как вскоре будет показано, CSS-класс представляет собой именованную коллекцию команд стилизации.

Команды стилизации имеют следующий формат:

```
имя_свойства: значение;
```

Каждый селектор может ссылаться на несколько команд стилизации. Каждая команда стилизации завершается точкой с запятой (`;`). Последняя команда, завершающаяся без точки с запятой, зачастую тоже воспринимается браузерами и задаваемый ею стиль выводится корректно. Тем не менее в любом случае предпочтительнее все же ставить точку с запятой в конце каждой команды стилизации. Для отделения последовательных команд стилизации друг от друга можно воспользоваться пробельными символами. Эти символы не влияют на механизм синтаксического разбора таблицы стилей браузерами, но оказывают существенную помощь при чтении содержимого таблиц стилей.

В стандарте CSS определен длинный перечень имен свойств, используемых для команд стилизации, и связанных с ними допустимых значений. Кроме того, в некоторых браузерах определяются нестандартные свойства, что позволяет добавлять собственные функциональные возможности.

В данном разделе дается краткий обзор назначения и синтаксиса селекторов. Затем мы приступим к исследованию CSS-свойств и их значений.

ПРИМЕЧАНИЕ

Контенты внедренного элемента style и файла таблицы стилей выглядят одинаково. Чтобы получить тот же самый графический результат, можно в любой момент взять содержимое внедренного элемента style и сохранить его в CSS-файле, на который дать ссылку.

Ссылка на элементы по их идентификаторам

В HTML все элементы на странице могут (а иногда и должны) иметь уникальный идентификатор. Наличие идентификатора не является обязательным, но его использование рекомендуется разработчикам страниц для однозначной идентификации заданного элемента. В HTML уникальный идентификатор задается элементу путем установки атрибута `id`:

```
<div id="header" ... />
```

У нескольких элементов одной и той же страницы не может быть одинаковых идентификаторов, хотя к этому положению браузеры иногда относятся весьма снисходительно и в случае его нарушения обычно не препятствуют выводу страницы на экран.

Образец ссылки на элемент по его идентификатору в таблице стилей имеет следующий вид:

```
#id {
   ...
}
```

Селектор состоит из строки идентификатора с префиксом, роль которого играет символ решетки `#`. Рассмотрим следующий HTML-элемент, визуализирующий кнопку:

```
<input id="button1" type="button" value="Say Hello!" />
```

Для стилизации кнопки с приданием ее надписи красного цвета нужно иметь в таблице стилей следующую команду:

```
#button1 {
   color: red;
}
```

Обычно ссылка на элемент по его идентификатору делается в том случае, когда вы не собираетесь многократно использовать команды стилизации для других элементов на страницах вашего сайта. Таким образом, применение селекторов на основе идентификаторов является инструментом стилизации одного конкретного элемента.

Ссылка на элементы по их именам

HTML-элементы стилизуются браузером автоматически, и каждый браузер может стилизовать их по-разному. Например, обычная кнопка стилизуется на iPhone со скругленными углами, а на Windows Phone — с прямыми.

CSS-стиль позволяет изменять внешний вид стандартных HTML-элементов путем создания селектора с именем тега элемента. Например, чтобы одинаково стилизовать все элементы гиперссылок (в данном случае — это элемент `a`), нужно определить следующий селектор:

```
/* все элементы получат желтый цвет и не будут выводиться с подчеркиванием */
 a {
   color: yellow;
   text-decoration: none;
}
```

Учтите, что в целом стилизация по именам тегов может порой оказаться чрезмерной, особенно это касается определенных тегов. Например, одинаковый внешний вид для всех ссылок более приемлем, чем для всех полей ввода или элементов таблицы на всех страницах сайта.

Ссылка на элементы с помощью собственных классов

Если одинаковый внешний вид нужно придать какому-то множеству элементов на одной или нескольких страницах, следует создать CSS-класс. Как уже упоминалось, такой класс является именованной коллекцией команд стилизации.

Класс определяется в таблице стилей путем указания произвольного имени класса с использованием префикса в виде символа точки (`.`). Рассмотрим пример:

```
/* Этот CSS-класс по имени "red-button" определяет кнопку с красным текстом */
 .red-button {
   color: red;
}
```

В исходном коде HTML-страницы CSS-класс присваивается элементу с помощью атрибута `class`. Рассмотрим пример:

```
<input id="button1" class="red-button" type="button" value="Say Hello!" />
```

А что, если на HTML-странице элементу будет присвоено имя несуществующего класса? Ничего страшного не произойдет, браузер просто проигнорирует эту команду.

Каскадная модель

При выводе на экран любого HTML-элемента браузер рассчитывает найти значение, определенное для каждого возможного и распознаваемого им CSS-свойства. Но это

не означает, что разработчики должны присвоить значение каждому возможному CSS-свойству каждого элемента страницы. Буква «C» в акрониме CSS говорит сама за себя. «C» означает *каскадная* (cascading), и смысл этого слова заключается в том, что браузеры предоставляют для каждого свойства значение, предлагаемое по умолчанию. Это значение может быть заменено разработчиком.

Фактическое значение каждого CSS-свойства, с помощью которого определяется порядок вывода элемента на экран, задается значением, которое можно присвоить непосредственно элементу или его самому дальнему родительскому элементу. В основе CSS-иерархии находится значение, присваиваемое браузером по умолчанию. Рассмотрим следующий фрагмент HTML-разметки:

```
<html>
   <body>
     <header>
       <div>
         Hello, world!
       </div>
       <div>
         This is me.
       </div>
     </header>
     This is the body of the page.
   </body>
</html>
```

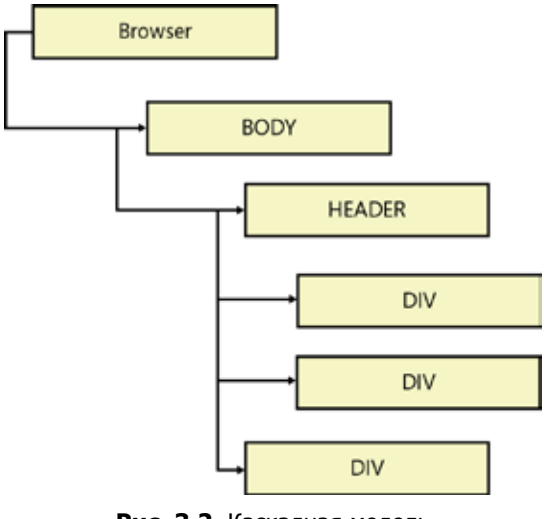

Рис. 3.2. Каскадная модель

Браузер назначает тексту исходную комбинацию шрифтовых и цветовых параметров. Вы можете переопределить эти параметры, назначив стиль элементу `body`. Если произвести такое переопределение, все параметры распространятся вниз

по дереву документа и будут применены к любому тексту страницы. Если вы намерены изменить параметры для текста в элементе `header` или всего лишь в одном из элементов `div` внутри заголовка, команду стилизации нужно добавить только к этому элементу. Любые параметры, которые явным образом не переопределены, сохраняют значение, присвоенное на более высоком уровне иерархии. Рисунок 3.2 иллюстрирует каскадную CSS-модель — в этой модели параметры, применяемые к тому или иному элементу, воздействуют на все элементы, находящиеся в поддереве, для которого этот элемент является корневым.

Основные команды стилизации

В синтаксис CSS включен ряд свойств, используемых для определения практически каждого аспекта HTML-элементов: шрифта, цветового оформления, размера, расположения, наличия тени и многого другого. Рассмотрим краткую сводку наиболее часто используемых свойств. Если вам нужна подробная справка, обратитесь по адресу http://bit.ly/Snr6cX.

Настройка цветового оформления

Изменение цветового оформления какого-либо текста зачастую становится первой попыткой, предпринимаемой разработчиками с целью взять под контроль функциональные возможности программы. Давайте рассмотрим варианты, доступные для изменения цветовых параметров в таблице стилей.

Определение цвета

Во всех встречавшихся до сих пор примерах ссылки на цвета давались по их названиям, например `black`, `red` или `white`. CSS позволяет выражать цвет другими, более эффективными способами. В стандартах HTML и CSS определено 147 названий цвета, среди которых можно найти такие названия, как `black`, `blue`, `fuchsia`, `gray`, `green`, `purple`, `red`, `silver`, `white` и `yellow`. Полный перечень предопределенных цветовых значений можно найти по ссылке http://bit.ly/QyqKUx.

В общем, цвета выражаются в виде трех целочисленных значений в диапазоне от 0 до 255, относящихся к их красному, зеленому и синему (Red, Green и Blue, RGB) компонентам. Для задания своего цвета можно воспользоваться любым из следующих вариантов синтаксиса:

```
.my-classic-button {
   color: #ff0000;
 }
 .my-fancy-button {
   color: rgb(255, 0, 0);
}
```

В первом примере для RGB-значений использовалось шестнадцатеричное выражение. Перед шестнадцатеричными значениями должен ставиться префикс #. Первые два символа относятся к красному, вторые — к зеленому, а последние два — к синему компоненту цвета.

В последнем примере вместо значения цвета используется функция rgb, которой передаются десятичные значения. В данном случае порядок цветовых компонентов также задается в виде значений для красной, зеленой и синей составляющих.

ПРИМЕЧАНИЕ

Браузеры позволяют назначать цвету уровень прозрачности. Для этого к определению цвета добавляется четвертое значение — альфа-канал. В шестнадцатеричном формате значение альфа задается в диапазоне от 0 (полная непрозрачность) до 255 (полная прозрачность). В RGB-формате значение альфа может быть выражено десятичным числом (например, 0,5) или процентным отношением (например, 50 %). При добавлении к цвету прозрачности следует воспользоваться функцией rgba:

```
.my-classic-button-transparent {
color: #88ff0000; /* прозрачность задается в первом значении (88) */
}
.my-fancy-button-transparent {
color: rgba(255, 0, 0, 0.5); /* прозрачность задается в четвертом значении (0.5) */
}
```

Следует иметь в виду, что устаревшие браузеры прозрачность при задании цвета могут не поддерживать. Однако для читателей данной книги, чей интерес проявляется в основном к программированию для Windows 8 и в контексте работы под управлением Windows 8, прозрачность при задании цвета полностью поддерживается.

Изменение цвета первого плана

Цвет любого текстового фрагмента, имеющегося на HTML-странице, управляется свойством color. Устанавливая значение для свойства color элемента body, вы придаете всему тексту страницы выбранный вами цвет, предлагаемый по умолчанию:

```
body {
    color: black;
}
```

Но одним цветом на странице можно не ограничиваться, цвет первого плана можно установить для любого элемента на странице. Для выбора одного или нескольких элементов используются селекторы в виде идентификаторов или классов. Например, ранее упоминавшийся класс red-button задает красный цвет для текста всех элементов, к которым он применим:

```
.red-button {
    color: red;
}
```

Изменение цвета фона

В HTML-дизайне цвет фона HTML-элемента играет важную роль, поскольку он позволяет добиться впечатляющих визуальных эффектов. У каждого элемента может быть собственный фон, нарисованный чистым цветом или цветовым градиентом. У него также может быть фон, текстурированный с помощью растрового изображения. Начнем с простейшего случая — чистого цвета. Для этого нужен следующий код:

```
.blue-button {
   color: #ffffff; /* белый */
   background-color: #0000ff; /* синий */
}
```

Свойство `background-color` получает цветовое выражение.

Использование градиентов

Для задания градиента фону HTML-элемента свойству `background` присваивается выражение, в котором дается описание типа желаемого градиента. Линейный вертикальный градиент, начинающийся с синего и заканчивающийся красным цветом, имея посредине белый цвет, задается следующим кодом:

```
.blue-button {
   color: #ffffff; /* white */
   background: linear-gradient(to bottom, blue, white 80%, red);
}
```

Если говорить точнее, белый цвет появляется ближе к завершению градиента (80 %), а красный занимает остальные 20 % фона. Ключевое слово `bottom` указывает на направление градиента (в данном случае — вниз). Аналогичным образом с помощью функции `radial-gradient` можно также создать радиальный градиент.

Использование для фона растрового изображения

Чтобы использовать в качестве фона элемента растровое изображение, нужно изучить еще несколько свойств. Рассмотрим следующий пример:

```
.img-button {
   color: #ffffff; /* белый */
   background-image: url(/images/button-bkgnd.png);
   background-repeat: no-repeat;
}
```

Свойство `background-image` позволяет сослаться на изображение, используемое в качестве фона. Как показано в примере, ссылка осуществляется с помощью функции `url`. Если изображение слишком мало для области фона, можно выбрать его повторение по вертикали и (или) по горизонтали, а также однократный вывод

на экран. Этот режим управляется через свойство `background-repeat` со значениями `no-repeat` (без повторения), `repeat-x` (для повторения по горизонтали) и `repeat-y` (для повторения по вертикали).

Растровые изображения часто используются для рисования фона всей страницы. А если на странице большой объем контента и она прокручивается горизонтально или вертикально, что тогда должно быть с изображением? Должно ли оно прокручиваться вместе со страницей или его нужно зафиксировать на исходной позиции, а контент прокручивать на его фоне? Всем этим можно управлять следующим образом:

```
body {
    color: #ffffff; /* white */
    background-image: url(/images/bkgnd.png);
    background-repeat: no-repeat;
    background-attachment: fixed;
}
```

Свойство `background-attachment` получает одно из двух значений: `fixed` (фиксированное) или `scroll` (с прокруткой), смысл которых понять несложно.

Управление текстом

В любом HTML-документе наиболее важной частью является текст. Следовательно, выбор правильного сочетания шрифта и эффектов является ключевым для успеха страницы в Сети. Все браузеры используют для текста шрифт, предлагаемый по умолчанию, но этот шрифт (Times New Roman) большинству пользователей не нравится. Переключиться на другое семейство шрифтом легче легкого, но сначала следует коснуться некоторых особенностей программирования для Windows 8.

ВНИМАНИЕ

В Windows 8 есть собственный весьма специфический пользовательский интерфейс, основным шрифтом в котором является Segoe UI. Поскольку приложения для Windows 8 можно создавать с помощью HTML и CSS, появляется возможность изменять шрифт. В некоторых случаях изменение семейства и размера шрифта может даже создать трудности, связанные с выкладыванием завершенного приложения в Магазин Windows. В общем, когда дело касается использования CSS для приложений Windows 8, изменение семейства шрифтов и размера шрифта становится непростой задачей. Без особой необходимости этого делать не нужно, но даже при наличии такой возможности задействовать ее нужно только для небольших фрагментов пользовательского интерфейса. Вместо этого требуется воспользоваться применяемыми в Windows по умолчанию таблицами JavaScript-стилей, с чем вы столкнетесь в главе 5.

Выбор семейства шрифтов и размера шрифта

В CSS для указания одного или нескольких семейств шрифтов служит свойство `font-family`. Когда речь заходит о шрифтах и браузерах, нужно понимать, что значение свойства `font-family` является всего лишь рекомендацией для браузера. Поскольку веб-страница находится на веб-сайте и просматривается на локальном компьютере, может оказаться, что локальный компьютер не оснащен запрашиваемым шрифтом. Поэтому лучше всегда дополнительно указывать альтернативные шрифты, как показано в следующем примере:

```
body {
   color: #ffffff; /* белый */
   background-image: url(/images/bkgnd.png);
   background-repeat: no-repeat;
   background-attachment: fixed;
   font-family: "trebuchet ms", helvetica, sans-serif;
}
```

Семейства шрифтов перечисляются в порядке предпочтения. Если недоступно первое указанное семейство, браузер переходит ко второму, и т. д. Если в имени шрифта имеются пробелы, то будет лучше заключить такое имя в кавычки даже при том, что на всех браузерах строгой необходимости в этом нет.

За многие годы у HTML-разработчиков сложилось мнение, что наилучшим выбором, по крайней мере, достойно визуализируемым на всех браузерах, являются шрифты Helvetica и Sans-serif.

А как насчет размеров шрифта? Нужный размер можно указать через относительную и абсолютную величину, использовав несколько единиц измерения. Например, можно указать такие ключевые слова, как `x-small`, `small`, `medium`, `large` и `x-large`. Эти ключевые слова имеют отношение к стандартному размеру шрифта браузера, измененному пользователем. Если указывать размер шрифта в процентном отношении, то реальный размер шрифта задается относительно окружающего текста. Если в качестве единицы измерения (`em`) используются числа в диапазоне от 0 до 1, то фактический размер задается относительно родительского элемента. И наконец, если задается количество пикселов (единица измерения `px`), то фактический размер определяется относительно разрешения экрана. Рассмотрим ряд примеров:

```
#footer {
   font-size: 80%;
 }
 #copyright {
   font-size: 9px;
 }
 #trademark {
   font-size: .8em;
}
```

С точки зрения вывода на экран больше подходит относительное задание размера. Абсолютное задание размера, например в пунктах (единица измерения `pt`) или, возможно, в миллиметрах (единица измерения `mm`), может привести к тому, что текст на разных компьютерах будет крупнее или мельче.

ВНИМАНИЕ

Как уже упоминалось, в приложениях для Windows 8 предпочтительнее задавать шрифты посредством определения классов в стандартных таблицах стилей Windows 8.

Стилизация шрифтов

Чаще всего исходный стиль шрифта меняется для придания какому-нибудь тексту особой важности. Это изменение означает изменение начертания шрифта этого текста на курсивное, полужирное или подчеркнутое. Жирность выражается через свойство `font-weight`, а начертание (например, курсивное) требует использования свойства `font-style`:

```
#footer {
   font-size: 80%;
   font-weight: 700;
   font-style: italic, underline;
}
```

Если нужно вывести часть текста полужирным шрифтом, можно просто установить для свойства `font-weight` значение `bold`. Если нужно управлять жирностью текста более тонко, можно выбрать значение в диапазоне от 400 (нормальное начертание) до 700 (жирное начертание). Значения выше или ниже крайних значений просто увеличивают или уменьшают степень «жирности» шрифта.

Режимы отображения HTML-страниц

HTML-страница создается как результат сочетания множества элементов. А как эти элементы реально объединяются браузером? Будут ли они составляться горизонтально или вертикально? Будет ли между ними задано пустое пространство? В этом разделе дается ответ на первый вопрос, а в следующем — на второй.

В HTML разрешены два основных режима отображения: линейный и блочный. Все HTML-элементы по умолчанию выводятся в одном из этих режимов. Например, элемент `div` всегда выводится как блок, а элемент `span` — линейно. Однако с помощью CSS режим вывода можно изменить. Свойство, позволяющее это сделать, называется `display`.

Блочные элементы

Блочные элементы выстраиваются относительно друг друга вертикально. Каждый блочный элемент занимает всю доступную высоту и визуализируется так, будто до него и после него стоит разрыв строки. Это означает, что браузер, столкнувшись с блочным элементом, начинает его вывод на экран с новой строки. Закончив вывод, он перед работой с другим элементом переходит на следующую строку. Визуализация заданного CSS-класса в качестве блока обеспечивается следующим кодом:

```
.headline {
   display: block;
}
```

Обычными HTML-элементами, по умолчанию визуализируемыми на экране в виде блоков, являются `h1` (и другие элементы заголовков), `div`, `p`, `ul`, `li`, `table` и `form`.

Линейные элементы

В отличие от блочных элементов, линейные элементы не заставляют браузер прерывать поток HTML-разметки при выводе на экран. Линейный элемент просто визуализируется на экран бок о бок с существующим контентом и занимает строго необходимое ему пространство. Вывод заданного CSS-класса в качестве линейного элемента обеспечивается следующим кодом:

```
.headline {
   display: inline;
}
```

Обычными элементами, выводимыми по умолчанию на экран в линейном режиме, являются `span`, `a`, `input` и `img`.

ПРИМЕЧАНИЕ

Используя свойство display, можно превращать блочные элементы в линейные и наоборот. То, что браузеры делают в обычном порядке, просто является предопределенным режимом, который может быть изменен разработчиком.

Обтекание элементов

Иногда одних только блочных или линейных режимов отображения для достижения ваших целей может оказаться недостаточно. Рассмотрим следующую HTML-разметку:

```
<div id="article">
   <img src="picture.png" />
   <span>Some possibly long text</span>
</div>
```

По умолчанию, изображение и текст визуализируются бок о бок, в результате текст, как показано на рис. 3.3, располагается на уровне нижней части изображения.

Рис. 3.3. Обычное выравнивание изображения и текста

С помощью CSS можно изменить вертикальное выравнивание текста так, чтобы он визуализировался на уровне верхней части или середины изображения. Однако ни один из этих приемов, наверное, не даст вам желаемый результат, представленный на рис. 3.4, — режим обтекания текста вокруг изображения.

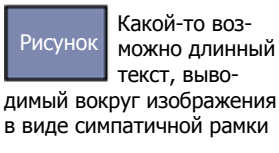

Рис. 3.4. Обтекание текста вокруг изображения

Для достижения этого эффекта вам нужно воспользоваться свойством `float`. Это свойство создает эффект визуализации элементов, следующих за плавающими элементами в непрерывном потоке, с автоматическим переходом на следующую строку при подходе к краю содержащего их блока. Чтобы обозначить направление обтекания для свойства `float`, используется значение `left` или `right`. Следующий CSS-код, примененный к предыдущему фрагменту HTML-кода, приводит к тому, что экран выглядит так, как показано на рис. 3.4.

```
#article {
   float: left;
   width: 100px;
}
```

В режиме обтекания элементы, следующие за плавающим элементом, поднимаются вверх, чтобы заполнить все доступное пространство, как при удалении в текстовом редакторе символов перевода строки. Свойство `float` аннулирует обычные режимы блочности и линейности. Однако в определенных местах приходится возвращаться к исходному положению вещей и останавливать обтекание. Элемент, на котором останавливается обтекание, может получить следующее стилевое оформление:

```
#stopFloating {
   clear: both;
}
```

Элемент, на котором останавливается обтекание, может быть настоящим элементом страницы (например, текстовым абзацем) или пустым элементом `div`, используемым только для отключения режима обтекания.

ПРИМЕЧАНИЕ

Если вы все еще не понимаете, чем отличается линейный элемент от плавающего, представьте себе, что линейность заключается в совокупности отдельных элементов, выложенных бок о бок на одной логической линии. А плавающими являются элементы, которые обтекаются другими элементами от начальной до конечной точки.

Блочно-линейные элементы

Обтекающие элементы вполне справляются с поставленной перед ними задачей, но и у них проявляются недостатки, когда они распространяются на несколько строк, причем у каждого элемента имеется свой размер, отличающийся от размера других элементов. При возникновении данной проблемы на выручку приходит третье допустимое значение свойства `display` — значение `inline-block`.

Элемент со значением `inline-block` является блочным элементом, выводимым на экран линейно вместе с другими блочными элементами. Внутри каждый блочный элемент выводится на экран как обычно со всеми необходимыми ему параметрами блочности, линейности и обтекания. Рассмотрим следующий фрагмент HTML-кода:

```
<ul id="container">
   <li>Block #1</li>
   <li>Block #2</li>
   <li>Block #3</li>
</ul>
```

А теперь стилизуем элементы:

```
li {
   display: inline-block;
   width: 100px;
   min-height: 100px;
   background-color: green;
   color: white;
   vertical-align: top;
}
```

Результаты показаны на рис. 3.5.

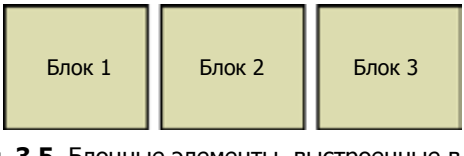

Рис. 3.5. Блочные элементы, выстроенные в ряд

Следует заметить, блочно-линейные элементы, выровненные по верхнему краю, также неплохо работают при наличии нескольких строк различной высоты. Если не применять обтекание, браузеры всегда используют для их визуализации одну связующую горизонтальную прямую.

Недостаток блочно-линейных элементов состоит в том, что они воспринимают все, что имеется в исходном HTML-коде, буквально. То есть пустые места, имеющиеся в исходном тексте (в случае фиксированной ширины страниц), могут стать причиной нежелательных переходов на следующую строку. Чтобы обезопасить себя от этого, нужно подумать о возможности написания своего HTML-кода одной строкой, без дополнительных пустых мест или переводов строки. Это может понадобиться только для той части страницы, в которой стиль элементов является блочно-линейным.

ПРИМЕЧАНИЕ

Как можно предположить, глядя на рис. 3.5, блочно-линейные элементы могут пригодиться для организации стиля пользовательского интерфейса Windows 8 с помощью HTML и CSS.

Разрядка и блочная модель

CSS позволяет создать вокруг каждого HTML-элемента относительно обширную и четко выраженную инфраструктуру. Эта инфраструктура, известная как блочная модель, показана на рис. 3.6.

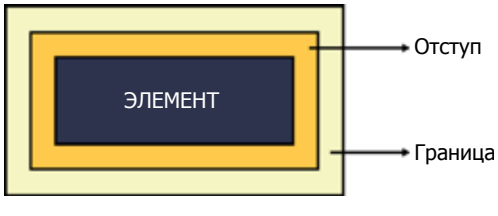

Рис. 3.6. Блочная CSS-модель

Блочная модель определяет то, что браузер ожидает встретить вокруг каждого отдельного HTML-элемента. Первый из окружающих блоков является блоком отступа и управляется с помощью свойства `padding`. Следующим является блок границ, используемых для обрамления элемента; параметры границы задаются с помощью свойства `border`.

Кроме того, каждый элемент может быть помещен в другой более крупный по размеру блок, известный как *блок полей* (margin box). Поля управляются свойством `margin` и определяют расстояние между элементом, помещенным в блок, и его соседями.

Определение отступов

Свойство `padding` определяет количество пикселов вокруг элемента. Дополнительные пикселы наследуют параметры фона элемента. Обычно отступы служат для добавления некоторого пространства вокруг контента (то есть текста) элемента, как показано на рис. 3.7.

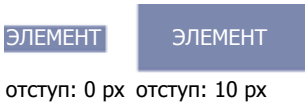

Рис. 3.7. Эффект от назначения отступа HTML-элементу

Если установить некое значение для свойства `padding`, то отступы будут одинаковыми вокруг всего элемента. Но вы можете устанавливать отступы отдельно для каждой из сторон элемента. Это делается с помощью свойств `padding-top`, `padding-left`, `padding-bottom` и `padding-right`.

Определение границ

Границами HTML-элемента можно управлять с помощью свойства `border`. Если установить это свойство, каждая из сторон элемента будет стилизована одинаково. Для настройки каждой из сторон по отдельности служат специальные подчиненные свойства `border-top`, `border-left`, `border-bottom` и `border-right`.

При стилизации границы можно указать ширину, цвет, радиус и стиль. Каждый из этих свойств имеет собственное подчиненное свойство, например `border-width`, `border-color`, `border-radius` и `border-style`. Ширину и радиус (для скругленных углов) задают в пикселах, а для выражения цветовых параметров служат цветовые CSS-строки. Рассмотрим пример:

```
.rounded-button {
   color: #ffffff; /* белый */
   background: linear-gradient(to bottom, blue, white 80%, red);
   border-radius: 5px;
   border-width: 1px;
   border-color: #ffffff; /* белый */
}
```

Граница элемента может быть сплошной (solid), точечной (dotted) или пунктирной (dashed). Соответственно, возможными значениями подчиненного свойства `border-style` являются `solid`, `dotted` и `dashed`.

Определение полей

В HTML полем называется расстояние между заданным элементом и его соседями. В отличие от отступа, пространство, указанное в качестве поля, не визуализируется

как часть элемента. Эта область выводится на экран с любым исходным контентом, принадлежащим родительским элементам.

Поля задаются в пикселах, и их можно задать независимо через обычные подчиненные свойства: `margin-top`, `margin-left`, `margin-bottom` и `margin-right`. Например, если установить для свойства `margin-bottom` значение `10px`, высота элемента увеличится на 10 пикселов. Эти лишние 10 пикселов высоты визуализируются на экране с прозрачным фоном.

Определение ширины и высоты

По умолчанию HTML-элементы занимают минимум пространства, необходимого для вывода их контента. В некоторых случаях может понадобиться присвоить им фиксированный размер в абсолютном или относительном измерении. Для этого понадобятся свойства `width` и `height`. Обычно значения для `width` и `height` устанавливаются в пикселах или процентах:

```
body {
   width: 100%;
}
```

А вот более интересный пример CSS-параметров, с помощью которых получается разметка, показанная на рис. 3.8. Начнем с HTML-кода:

```
<body>
   <header>
     Header of the page
   </header>
   <div id="main">
     main
   </div>
</body>
```

А вот связанный с этим кодом CSS-код:

```
body {
   margin: 0px;
 }
 header {
   width: 100%;
   height: 80px;
 }
 #main {
   width: 930px;
   height: 100%;
   margin: 0px;
   margin-left: auto;
}
```

Поле элемента `body` установлено в 0 пикселов, следовательно, между контентом страницы и краями окна браузера пустого пространства не будет. Элемент `header`

займет всю ширину экрана и только 80 пикселов высоты. Элемент `div` с идентификатором `main` получит полную (оставшуюся) высоту экрана и будет органичен фиксированной шириной в 930 пикселов. Но что еще интереснее, поле элемента `div` сначала устанавливается в 0, а затем значение для левого поля заменяется специальным значением `auto`, означающим, что поле подбирается браузером автоматически, чтобы контент оказался в центре.

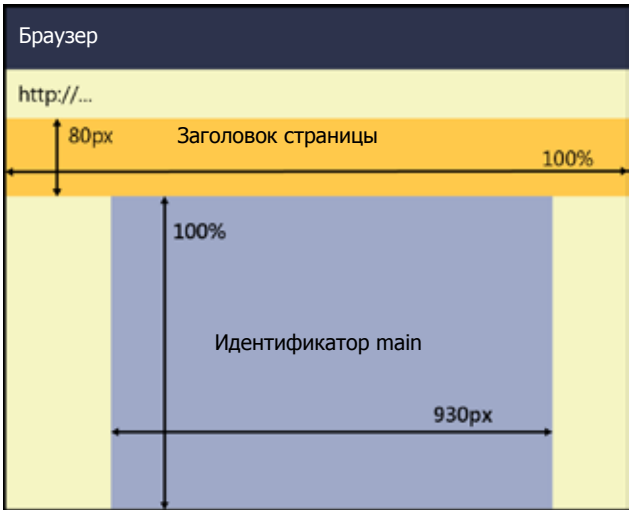

Рис. 3.8. Свойства width и height служат для создания неплохого (и весьма распространенного) макета страницы

ПРИМЕЧАНИЕ

Что касается ширины и высоты, то для них также имеются подчиненные свойства для определения минимальной и максимальной ширины или высоты. Этими свойствами являются min-height, max-height, min-width и max-width. Следует также отметить, что свойства width и height относятся к размеру контента, а для отступов, границ и полей они не учитываются.

Нетривиальное применение CSS

До сих пор вам попадались только основные CSS-селекторы — идентификаторы, имена тегов и создаваемые классы. Однако синтаксис CSS-селекторов намного богаче и сложнее. В нем можно сочетать множество встроенных псевдоклассов и операторов, определяя своего рода мощный язык запросов. Таким образом можно выбирать весьма специфический набор стилизуемых HTML-элементов.

CSS-псевдоклассы

CSS-псевдоклассы служат для наделения некоторых селекторов специальной возможностью отправки запросов:

```
селектор:псевдокласс {
   ...
}
```

Давайте рассмотрим ряд примеров часто применяемых псевдоклассов.

Псевдоклассы указателей

Коротко говоря, псевдоклассы похожи на хитрые операторы запросов, которые можно применять в отношении конкретных HTML-элементов, чтобы еще больше сузить область использования стиля. Некоторые псевдоклассы являются обобщенными и могут применяться практически ко всем селекторам, а некоторые, напротив, имеют смысл применительно лишь к вполне определенным элементам. Давайте рассмотрим псевдоклассы, работающие с указателями.

ПРИМЕЧАНИЕ

Указатели являются одними из основных HTML-элементов. Указатель (anchor) — своего рода техническое название гиперссылки, а для идентификации элемента используется буква «a». Указатель состоит из двух основных частей: визуализируемого текста и положенного в его основу URL-адреса. Встречая указатель, браузеры обычно выводят визуализируемый текст подчеркнутым, а заданный URL-адрес показывают в строке состояния. Когда пользователь щелкает на указателе, браузер переходит на заданный URL-адрес.

Обычно браузер выводит на экран каждый элемент `a` по-разному, в зависимости от его состояния (переход по ссылке уже был, ссылка активна, на ссылку наведен указатель мыши). Каждое состояние отображается на свой псевдокласс. Полностью стилевое оформление указателя выглядит следующим образом:

```
a {
   text-decoration: none; /* подчеркивание отсутствует */
   color: #aaaa66;
   padding: 0px;
 }
 a:visited {
   text-decoration: none; /* подчеркивание отсутствует */
   color: #aaaa66;
 }
 a:hover {
   text-decoration: underline;
   background-color: orange;
   color: black;
   padding: 2px;
 }
```

```
 a:active {
   text-decoration: none; /* подчеркивание отсутствует */
   color: #aaaa66;
}
```

Псевдокласс `visited` определяет внешний вид указателя, на котором был хотя бы один щелчок. Псевдокласс `hover` относится к внешнему виду в момент наведения указателя мыши. И наконец, псевдокласс `active` относится к внешнему виду указателя, когда он находится в фокусе ввода. В предыдущем примере вид указателя на экране не меняется, а другую цветовую схему он получает при наведении на него указателя мыши.

Псевдоклассы для элементов ввода

Иногда требуется изменить стиль элементов управления (то есть текстовых блоков), которые недоступны, помечены или имеют фокус ввода. Эта задача облегчается наличием в CSS ряда специализированных псевдоклассов. Псевдокласс `checked` служит для пометки всех установленных флажков или переключателей. А псевдоклассы `disabled` и `enabled` можно использовать для выбора недоступных или доступных для ввода элементов управления. Псевдокласс `focus` сигнализирует о том, какой из элементов в данный момент имеет фокус ввода.

Код следующего примера автоматически переключает цвет фона текстового поля `input` для индикации состояния наличия фокуса ввода:

```
input[type=text]:focus {
   background-color: orange;
   color: black;
}
```

Предыдущий фрагмент кода содержит интересную, ранее не встречавшуюся конструкцию: выражение в квадратных скобках. Это псевдокласс атрибутов.

Псевдоклассы атрибутов

Псевдоклассы атрибутов применяются к существующему CSS-селектору и ограничивают его визуализацию только теми элементами, в которых содержится заданный атрибут с заданным значением. Рассмотрим предыдущий фрагмент кода более подробно:

```
input[type=text]:focus {
   background-color: orange;
   color: black;
}
```

Селектор `[type=text]` применяется к элементу `input` и ограничивает перечень элементов `input` только теми из них, атрибут `type` которых имеет значение `text`. Наиболее популярные псевдоклассы атрибутов перечислены в табл. 3.1.

Таблица 3.1. Популярные псевдоклассы атрибутов

Селектор	Описание
[атрибут]	Элементы, содержащие атрибут с указанным именем
[атрибут = значение]	Элементы, содержащие атрибут с указанными именем и значением
[атрибут *= значение]	Элементы, содержащие атрибут с указанным именем, чье значение содержит указанное значение
[атрибут != значение]	Элементы, содержащие атрибут с указанным именем, чье значение отличается от указанного значения
[атрибут ^= значение]	Элементы, содержащие атрибут с указанным именем, чье значение начинается с указанного значения
[атрибут $= значение]	Элементы, содержащие атрибут с указанным именем, чье значение заканчивается указанным значением

Псевдоклассы для выбора дочерних элементов

Иногда нужно просто выбрать некоторые дочерние элементы для заданного селектора. Псевдокласс `first-child` выбирает первый дочерний элемент, а псевдокласс `last-child` — последний. Кроме того, можно выбрать *n*-й дочерний элемент заданного элемента, использовав псевдокласс `nth-child(`*n*`)`, где вместо *n* указывается индекс выбираемого дочернего элемента.

Составные селекторы

Как вы уже, наверное, догадались, CSS-селекторы могут образовывать комбинации для создания запросов любой рациональной сложности. Для определения запросов селекторы можно объединять с помощью пробела, запятой или других специальных символов. Наиболее распространенные случаи показаны в табл. 3.2.

Таблица 3.2. Наиболее популярные составные селекторы

Селектор	Пример	Описание
элемент элемент	div a	Выбор всех элементов a, содержащихся в элементах div
элемент, элемент	div,p	Выбор всех элементов div и всех элементов p
элемент > элемент	div > a	Выбор всех элементов a, являющихся прямыми потомками элемента div
элемент + элемент	div + a	Выбор всех элементов a, следующих непосредственно после элемента div

В дополнение рассмотрим следующий пример:

```
div.headline a {
    ...
}
```

Выражение `div.headline` указывает на все элементы `div` заданного CSS-класса по имени `headline`. А следующее за этим выражением уточнение сужает запрос до всех элементов `a`, содержащихся в любом поддереве, корневым для которого является элемент `div` с классом `headline`.

ПРИМЕЧАНИЕ

При использовании классов и псевдоклассов регистр символов не учитывается.

Медиазапросы

Когда CSS-файл привязывается к HTML-документу посредством элемента `link`, можно выбрать ту среду, в которой данный стиль будет применяться. Среда указывается посредством атрибута `media`. Если атрибут `media` не указан, браузер считает, что один и тот же CSS-файл должен использоваться при любых обстоятельствах.

Вначале роль атрибута `media` ограничивалась использованием разных CSS-файлов для просмотра и распечатки заданной HTML-страницы. Соответственно, первыми допустимыми значениями атрибута `media` были `screen`, `print` и `all`.

Самые современные браузеры поддерживают в атрибуте `media` настоящий язык запросов, позволяющий выбирать CSS-файл в зависимости от характеристик применяемого браузера.

Динамический выбор CSS-файла

Рассмотрим пример динамической привязки CSS-файлов к HTML-странице с помощью медиазапросов. Замысел состоит в том, что странице предлагается ряд вариантов и браузер динамически подбирает наиболее приемлемый из них, исходя из имеющейся среды:

```
<link type="text/css"
    rel="stylesheet"
    href="tiny.css"
    media="only screen and (max-width: 320px)">
 <link type="text/css"
    rel="stylesheet"
    href="large.css"
   media="only screen and (min-width: 321px)">
```

Что интересно, современные браузеры также могут выбирать CSS-файл, исходя из динамического изменения условий. Что касается представленного фрагмента

кода, он означает, что браузер автоматически применяет файл tiny.css, когда ширина экрана не превышает 320 пикселов, и переключается на файл large.css, когда размер окна увеличивается как минимум до 321 пиксела.

Механизм медиазапросов служит двум целям. С одной стороны, он дает возможность создавать страницы, адаптирующиеся под текущий размер экрана, а с другой — предоставляет простое, но эффективное средство придания вашим страницам мобильного интерфейса практически без особых усилий. В простейшем варианте мобильное устройство является, по сути, браузером с меньшим размером экрана.

Синтаксис медиазапросов

Как уже упоминалось, текущие условия работы браузера могут повлиять на выбор CSS-файла. Проведем краткий обзор информации о текущем состоянии, к которой имеется доступ для принятия решения о выборе CSS-файла. В первую очередь команда медиазапроса описывает, как он должен игнорироваться устаревшими браузерами. Основные свойства медиазапросов перечислены в табл. 3.3.

Таблица 3.3. Свойства наиболее популярных медиазапросов

Свойство	Описание
width	Возвращает ширину текущей области просмотра браузера. Это свойство поддерживает префиксы min и max
height	Возвращает высоту текущей области просмотра браузера. Это свойство поддерживает префиксы min и max
orientation	Возвращает значение portrait (книжная ориентация), когда значение свойства высоты больше значения свойства ширины или равно ему. В противном случае возвращает значение landscape (альбомная ориентация)

Свойство `orientation`, к примеру, позволяет определить (более точно, чем просто по ширине), какова ориентация страницы, книжная или альбомная.

Кроме того, для создания выражений любой сложности можно воспользоваться операторами `AND` и `NOT`.

Выводы

CSS — это язык, использующийся для добавления к HTML-страницам графических стилей. Он основан на наборе команд, выбирающих один или несколько HTML-элементов и изменяющих их вид, предлагаемый по умолчанию. Идентификация целевых элементов проводится посредством селекторов, а их стилизация — посредством присваиваний вида свойство-значение.

В этой главе вы узнали, как определять селекторы, и изучили наиболее важные свойства, которые вам, возможно, пригодятся для получения более впечатляющих результатов при разработке страниц.

Как разработчик, которого в первую очередь интересуют приложения для Windows 8 (в отличие от обычной веб-разработки), вы, наверное, в большинстве рабочих моментов будете пользоваться готовыми таблицами стилей и соглашениями, касающимися Windows 8. Но в любом случае материал данной главы должен стать основой для понимания любых функциональных возможностей CSS, которые будут встречаться на вашем пути.

Чтобы завершить вводную часть этой книги, касающуюся веб-разработки, давайте познакомимся с языком программирования JavaScript.

4 Знакомство с JavaScript

Ухвати суть вопроса, слова найдутся.

Катон-старший

У языка JavaScript, наверное, девять (или даже больше) жизней. Официальный дебют JavaScript состоялся в 1996 году в качестве составной части браузера Netscape, и он счастливо пережил первые 15 лет эпохи Интернета. Изначально предполагалось, что это будет простой язык, ориентированный на веб-разработчиков, желающих сделать свои HTML-страницы более восприимчивыми и привлекательными. JavaScript никогда не разрабатывался в качестве полноценного языка программирования, главный упор делался на простоту его применения.

И, в конце концов, он заработал очень хорошо.

У языка JavaScript удивительно низкий «барьер знакомства», но при этом он достаточно гибок, чтобы позволить опытным специалистам делать с его помощью почти все, что им вздумается. Сегодня знание JavaScript является одним из важнейших показателей мастерства практически всех разработчиков, однако, хотя освоить этот язык совсем несложно, добиться истинного мастерства в работе с ним не так-то просто.

С выпуском Windows 8, компания Microsoft сделала HTML5 и JavaScript первоклассной платформой для создания приложений. Разработчики используют HTML5 для создания разметки, CSS — для придания этой разметке стиля, а JavaScript — для манипулирования элементами страниц. И в то же самое время у разработчиков есть

доступ к целому набору системных библиотек, предлагающих виджеты и компоненты пользовательского интерфейса, позволяющие разработчикам обращаться к инструментам Windows 8.

Ожидается, что JavaScript-разработчики, пишущие приложения для Windows 8, в первую очередь заинтересованы в привязке загружаемых данных к элементам пользовательского интерфейса. Тем не менее это не означает, что JavaScript не может использоваться для реализации, по крайней мере, какой-то части бизнес-логики. Но, в общем, следует иметь в виду, что чем больше бизнес-логики вам придется реализовывать средствами JavaScript, тем, наверное, меньше JavaScript будет удовлетворять вас в качестве выбранного языка программирования.

Эта глава предназначена для того, чтобы освежить ваши знания основ языка JavaScript и рассмотреть ряд основных вариантов организации JavaScript-кода в приложении для Windows 8.

Основы языка

В основе языка JavaScript лежат жесткие стандарты, определенные в таких документах, как ECMA-262 и ISO/IEC 16262:2011. В отличие других популярных языков программирования, например C#, JavaScript-код не компилируется перед выполнением. Вместо этого JavaScript-код интерпретируется, а затем выполняется «на лету». Поэтому для запуска JavaScript-программ нужна среда времени выполнения (движок, транслирующий и исполняющий написанный вами код). На платформе Windows 8 язык JavaScript поддерживается средой времени выполнения; эта среда, называемая Chakra (произносится «Чакра»), подобно буддийской силе, в честь которой она названа, вдыхает жизненную энергию в тело HTML-страниц.

ПРИМЕЧАНИЕ

Основная масса языков программирования (таких, как C#, Visual Basic, C++ и Java) позволяет разработчикам выражать намеченное поведение с помощью высокоуровневого синтаксиса. Но код, выраженный таким образом, сначала должен быть «откомпилирован» в язык более низкого уровня, который значительно ближе к фактическому поведению машины. Иными словами, язык программирования, обычно используемый разработчиками, является простой абстракцией поведения машины. Классические компилируемые языки нуждаются в специальном инструментарии — компиляторе, чтобы трансформировать высокоуровневый синтаксис в низкоуровневый. Вы не можете запустить свою программу, пока она не будет успешно откомпилирована. Другие языки, такие как JavaScript, не являются компилируемыми. Но это не означает, что они не нуждаются для запуска в какой-либо адаптации; некомпилируемый (интерпретируемый) язык просто проходит свой «компилируемый» этап в процессе выполнения, непосредственно перед выполнением каждой конкретной строки кода.

Давайте теперь познакомимся с основами языка, в частности, с системой типов, переменными и функциями, а также с полезными и вредными привычками при программировании.

Система типов в JavaScript

В JavaScript система типов состоит из нескольких примитивных типов и нескольких встроенных объектов. Однако при написании JavaScript-кода диапазон типов, с которыми можно работать, оказывается гораздо шире. Вдобавок к встроенным объектам можно использовать объекты, предоставляемые хостом, а также объекты, импортируемые из сред, на которые имеются внешние ссылки. В случае приложений для Windows 8 нужно, в частности, импортировать библиотеку WinJs (Windows 8 JavaScript), которая является естественным шлюзом для прикладного программного интерфейса (Application Programming Interface, API) Windows 8.

Однако в этой главе основное внимание сосредоточено на исходной системе типов, соответствующей ранее упомянутым определениям стандарта JavaScript ECMA-262.

Примитивные типы и встроенные объекты

В JavaScript примитивными типами являются `number` (число), `string` (строка), `Boolean` (булево значение), `undefined` (неопределенное значение), `Object` (объект) и `Function` (функция). К встроенным объектам относятся `Array`, `Math`, `Date` и `RegExp`, плюс ряд объектов, которые являются всего лишь функционально обогащенными оболочками некоторых примитивных типов. Такие объекты-оболочки, называемые `String`, `Boolean` и `Number`, лишь добавляют дополнительные возможности соответствующим примитивным типам.

Тип `number` представляет собой числа с плавающей точкой, имеющие ноль и более десятичных знаков. Отдельных типов для таких вещей, как целые числа, длинные целые числа, числа с одинарной, двойной точностью или байты, не существует. Одно специальное число, `NaN`, зарезервировано для чисел, не имеющих математического смысла, и является переменной, содержащей результат математической операции, не имеющей смысла. Фактически, название `NaN` является акронимом от Not-A-Number (не число). Встроенный объект `Number` — это лишь оболочка примитивного числового значения, которая добавляет метод преобразования числа в строку.

Тип `string` представляет собой последовательность, состоящую из нуля и более символов. Содержимое строки заключается в соответствующие друг другу пары одинарных или двойных кавычек. Встроенный объект `String` добавляет ряд методов, включая `substring`, который извлекает часть строки между двумя указанными индексами, и `toLowerCase`, который переводит символы строки в нижний регистр.

Остальные встроенные объекты перечислены в табл. 4.1.

Таблица 4.1. Встроенные JavaScript-объекты

Встроенный объект	Описание
Array	Предлагает богатый программный интерфейс для коллекции JavaScript-объектов. Доступ к элементам можно получить по индексам, также можно добавлять новые элементы или удалять существующие
Date	Предлагает множество полезных методов для работы с датами, включая получение и установку отдельных элементов, таких как день, месяц или год. Также работает с данными времени
Math	Предлагает интерфейс для решения множества математических задач, включая получение случайных чисел, возведение в степень, округление в меньшую или большую сторону, получение абсолютного значения
RegExp	Предлагает регулярные выражения

Глобальный JavaScript-объект

Все JavaScript-объекты наследуют свойства уникального глобального суперобъекта, затем свойства и функции глобального объекта дополняют все собственные и создаваемые объекты. В частности, глобальный объект имеет следующие свойства:

- `NaN` — возвращает значение, не являющееся числом (Not-A-Number).
- `Infinity` — возвращает значение бесконечности.

В табл. 4.2 перечислены функции, доступные в глобальном объекте.

Таблица 4.2. Функции, доступные в глобальном объекте

Функция	Описание
eval	Получает код в виде строки и выполняет его «на лету»
escape/unescape	Кодирует и декодирует специальные символы в обычной строке
encodeUri/decodeUri	Кодирует и декодирует специальные символы в URI-строке
isNaN	Возвращает true, если указанное значение не относится к числу
isFinite	Возвращает true, если указанное значение является бесконечностью
parseInt	Проводит синтаксический разбор строки и пытается извлечь из нее целое число
parseFloat	Проводит синтаксический разбор строки и пытается извлечь из нее число с плавающей точкой

Следует отметить, что любой JavaScript-объект, значение или константа, казалось бы, не принадлежащий другому JavaScript-объекту, в конечном счете оказывается доступным, поскольку он предоставляется глобальным JavaScript-объектом.

Сравнение нулевых и неопределенных значений

В JavaScript нулевое (`null`) значение несколько отличается от такового во многих языках высокого уровня, таких как C# и Java. Как вскоре выяснится, JavaScript-переменная не привязана к какому-то конкретному типу. То есть переменная не привязана ни к какому типу до тех пор, пока ей явным образом не присвоено какое-нибудь значение. На данном этапе считается, что переменная не определена (undefined).

В JavaScript неопределенное значение рассматривается не как значение, а как специальный тип. Если с помощью оператора `typeof` проверить тип переменной, которой не присвоено значение, будет получена строка `undefined`. Если вместо этого попытаться вычислить содержимое такой переменной, будет получено нулевое (`null`) значение. Обратите внимание на следующий код:

```
var x;  // по этой причине переменная имеет тип undefined
var y = null;
```

А что произойдет при сравнении `x` и `y`?

В JavaScript имеется два оператора равенства: оператор двойного равенства (`==`) и оператор тройного равенства (`===`). Если используется оператор `==`, оба выражения вычисляются, а получившиеся значения сравниваются. Но если используется оператор `===`, сравниваются также и типы значений. Возвращаясь к предыдущему фрагменту кода, при сравнении `x` и `y` с помощью оператора `==` вы получите результат `true`, означающий, что неопределенное значение переменной `x` в конечном итоге вычисляется в `null`, что соответствует значению, присвоенному переменной `y`.

В отличие от этого при проведении операции сравнения `x` и `y` с помощью оператора `===` будет получено значение `false`, означающее, что обе переменные содержат одно и то же значение, но относящееся к разным типам.

Переменные

В JavaScript переменные являются всего лишь местами хранения данных и не имеют постоянно действующего ограничения на хранение значений фиксированного типа. Когда значение присвоено, переменные приобретают тип тех данных, которые в них хранятся. Поэтому, как показано в следующем фрагменте кода, JavaScript-переменная за время своего существования может менять свой тип несколько раз:

```
var data = "dino"; // сейчас данные имеют строковый тип
data = 123;        // сейчас данные имеют числовой тип
```

JavaScript-переменные начинают существовать после своего первого использования, до этого они содержат значение `null`, а их тип определяется как `undefined`.

Локальные переменные

При определении переменных следует всегда использовать ключевое слово `var`, которое является подсказкой как для системы синтаксического разбора (парсера), так и для вас. Тем не менее применять ключевое слово `var` не обязательно, а вот строго контролировать область видимости переменной входит в число настоятельных рекомендаций.

Переменные, определенные в теле функции, имеют область видимости внутри этой функции, относясь при этом к локальным переменным, только если они были определены с помощью ключевого слова `var`. Если ключевое слово `var` опустить, переменные будут считаться глобальными, но при этом оставаться неопределенными до тех пор, пока функция не будет выполнена хотя бы один раз. Следует также учесть, что в JavaScript отсутствует понятие области видимости на уровне блока, которое встречается во многих других языках программирования. Рассмотрим следующий код:

```
function foo(number) {
   var x = 0;        // Переменная x локальна по отношению
                     // к функции и вне ее не видна

   if (number >0) {
     var y = number; // Переменная y НЕ является локальной
                     // по оотношению к блоку IF
     ...
   }
}
```

Переменная `y` не является локальной по отношению к блоку `if`, то есть ее содержимое доступно также вне блока. Однако поскольку эта переменная определена с помощью ключевого слова `var`, она является локальной по отношению к функции. Важно отметить, что если не воспользоваться ключевым словом `var` в блоке `if`, то нужно быть готовым к тому, что эта, по сути, временная переменная будет возведена в более высокий ранг глобальной переменной!

Положение, только что показанное в отношении инструкции `if`, справедливо также и по отношению к циклам `for` и `while`.

Глобальные переменные

Переменные, объявленные в глобальной области видимости, то есть за пределами тела какой-либо функции, всегда являются глобальными независимо от того, используется ключевое слово `var` или нет. Рассмотрим следующий фрагмент кода, чтобы понять разницу между локальной и глобальной переменными, представленными с помощью ключевого слова `var`:

```
var rootServer = "http://www.expoware.org/"; // глобальная переменная
 section = "mobile";                         // глобальная переменная

 function doSomething() {
   var temp = 1;                             // локальная переменная
   mode = 0;                                 // глобальная, но
                                             // неопределенная переменная
}
```

JavaScript-среда времени выполнения хранит глобальные переменные как свойства скрытого объекта, на который можно сослаться с помощью ключевого слова `this`. Учтите, что браузеры часто создают зеркальный образ глобального объекта с помощью объекта `window`.

Защита глобального пространства имен

Почти в каждом языке программирования, если есть возможность использования глобальных переменных, работа существенно упрощается. Но у глобальных переменных есть свои недостатки. Важным недостатком является риск возникновения конфликта имен между переменными, определенными в различных частях вашего кода, в библиотеках сторонних производителей, в коде рекламодателей, в аналитических библиотеках. Конфликт имен в сочетании с динамической типизацией JavaScript-переменных может привести к непреднамеренному изменению состояния приложения с потенциально неприятными аномалиями во время его выполнения.

Рассмотрим, с какой легкостью можно невольно создать глобальные переменные: забудете поставить ключевое слово `var` — получите глобальную переменную; допустите опечатку в имени переменной при присваивании значения — получите совершенно новую глобальную переменную. Последний случай возможен благодаря тому, что в JavaScript разрешается использовать переменные без их предварительного объявления.

Но всего этого можно избежать, если действовать обходным путем. Глобальные переменные лучше всего создавать в качестве свойств объекта-оболочки. Код, похожий на следующий, помещается в JavaScript-файл, на который затем делается ссылка с каждой страницы:

```
var Globals = (function() { return this; }());
```

Затем делается ссылка на используемый глобальный объект с помощью выражения `Globals.`*Xxx*, где *Xxx* — имя нужной глобальной переменной. Таким образом все ваши глобальные переменные будут, по крайней мере, выделены в коде.

ПРИМЕЧАНИЕ

Места в вашем коде, выходящие за рамки установленных образцов, включая отсутствие ключевых слов var, помогает найти программа JSLint — интерактивное средство для статического анализа JavaScript-кода (http://www.jslint.com).

Переменные и их подъем

Подъем переменных (hoisting) — это такая функциональная возможность JavaScript, которая позволяет разработчикам объявлять переменные где угодно в области видимости, а затем повсеместно их использовать. В JavaScript можно сначала задействовать переменную, а затем, уже позже, ее объявить (например, с помощью ключевого `var`). В общем, программа ведет себя так, будто инструкция `var` находится в начале программы. Рассмотрим пример:

```
function() {
   mode = 1;
   ...
   var mode;
}
```

Исторически эта возможность была введена в язык, чтобы максимально упростить применение JavaScript неопытными разработчиками. Однако при написании больших объемов JavaScript-кода подъем переменных становится явным источником недоразумений и ошибок. Лучше взять за правило помещать все свои переменные в начало каждой функции, причем лучше в одну инструкцию `var`, как показано в следующем фрагменте:

```
function() {
   var start = 0,
   total = 10,
   index;
   ...
}
```

Хотя можно отметить, что наличие вместо одной нескольких инструкций `var` не считается чем-то предосудительным или неправильным, если постоянно придерживаться правила применения одной инструкции `var`, это заставит вас всегда определять переменную перед ее использованием.

Объекты

JavaScript обычно не относят к объектно-ориентированным языкам, по крайней мере, на таком уровне, как Java и C#. Главная причина кроется в том, что определение объекта в JavaScript отличается от общепринятого определения объекта в классических объектно-ориентированных языках.

Структура JavaScript-объектов

Объект в JavaScript является словарем, состоящим из пар имя-значение. Структура объекта скрыта, и у вас нет способа обращения к нему. JavaScript-объект обычно имеет только данные, но к нему можно также добавить поведение. Структура объекта (ее видимая часть) в любой момент может быть изменена: например, к ней

в процессе выполнения программы можно добавлять новые методы и свойства. Скрытая часть структуры никогда не меняется. Добавить свойство к существующему объекту можно следующим образом:

```
var theNumber = new Number();
theNumber.type = "Number";
```

Добавление члена к JavaScript-объекту работает только для данного конкретного экземпляра. Если бы вы сейчас создали еще один экземпляр объекта `Number`, свойство `type` для нового объекта `Number` было бы недоступно. Эту проблему можно обойти с помощью прототипов.

Общее свойство prototype

Чтобы добавить новый член ко всем созданным экземплярам данного типа, его нужно добавлять в прототип объекта. В следующем примере свойство `type` добавляется ко всем числам для получения большего контроля над действительным типом объекта:

```
if (typeof Number.prototype.type === 'undefined') {
   Number.prototype.type = "Number";
}
```

ВНИМАНИЕ

Этот пример, иллюстрирующий, в частности, применение собственного нестандартного свойства type, приведен здесь не случайно. Чтобы определить тип объекта, хранящегося в данный момент в переменной, в JavaScript можно воспользоваться оператором typeof, однако в большинстве случаев в качестве ответа вы получите лишь слово «Object». То есть простого способа определения различия между, к примеру, строками и числами, не существует. Поэтому умение использовать собственное нестандартное свойство type становится еще более важным.

Создание новых экземпляров объекта

Тип `Object` позволяет создавать из значений и методов агрегаты, являющиеся максимально возможным JavaScript-приближением к стандартной объектной модели, такой как, например, в C#. Новый объект можно создать следующим образом:

```
var dog = new Object();
dog.name = "Jerry Lee Esposito";
```

Обычно непосредственное использование конструктора `Object` не приветствуется. Лучше, как показано далее, задействовать литерал объекта:

```
var dog = {
   name: "Jerry Lee Esposito",
};
```

Использование конструктора `Object` становится причиной проблем производительности JavaScript-интерпретатора, которому приходится определять при вызове конструктора его область видимости и искать потенциально большое пространство стека. Кроме того, непосредственное использование конструктора не передает смысла объектов в качестве словарей, что является ключевым пунктом JavaScript-программирования.

Функции

В JavaScript *функцией* называют блок кода, которому дополнительно может быть присвоено имя. Если имя функции не дается, она называется *анонимной* (безымянной). Функции имеют область видимости и рассматриваются как объекты; у них могут быть свойства, и они могут передаваться в качестве аргументов, с которыми можно взаимодействовать.

Функции служат для решения двух основных задач: определения повторяющегося поведения и создания нестандартных объектов.

Именованные функции для определения повторяющегося поведения

Именованная функция (в отличие от безымянной) определяется следующим образом:

```
function doSomeCalculation(number) {
   ...
   return number;
}
```

Определенная таким образом функция имеет глобальную область видимости и интерпретируется как новый член, добавляемый к глобальному JavaScript-объекту. Функцию можно вызвать из любого места вашего кода следующим образом:

```
var result = doSomeCalculation(3);
```

JavaScript-функции можно вызывать с любым количеством аргументов независимо от количества объявленных параметров. Иными словами, у вас может быть функция `doSomeCalculation` (вычисли что-нибудь), для которой объявлен всего один аргумент (число), но ее можно вызвать, передав ей любое (большее) количество аргументов:

```
var result = doSomeCalculation(1, 2);
```

Хотя браузеры смотрят на такое программирование весьма снисходительно, хорошей эту практику не назовешь. Но такой гибкостью вполне можно воспользоваться для свободного объявления функций, которые допускают произвольное количество аргументов.

В JavaScript функция может получить доступ к объявленным параметрам по имени или по позиции, использовав предопределенный массив под названием `arguments`. Этот массив возвращает фактический перечень параметров, переданных функции. Таким образом, следующая функция может получать любое количество аргументов и заниматься их обработкой:

```
function doSum() {
   var result = 0;
   for(var i=0; i<arguments.length; i++)
     result += arguments[i];
   return result;
}
```

Для функции `doSum` допустимы оба этих вызова:

```
var r1 = doSum(1, 2, 3);
var r2 = doSum(4, 5);
```

Функцию, принимающую произвольное количество аргументов, наверное, более четко можно определить как функцию, не имеющую явных формальных параметров. Но у вас также может быть несколько объявленных параметров, при этом функция все равно будет принимать и обрабатывать любое количество параметров. Во всяком случае, массив `arguments` возвращает фактически перечень параметров, найденных в вызове функции, в том порядке, в котором они перечислены.

Функции и объекты немедленного вызова

В JavaScript *функция немедленного вызова* (immediate function) представляет собой фрагмент кода, определяемый и выполняемый в одном и том же месте. Обычно определения функций помещают в одно место (скажем, в отдельный файл), а их вызов происходит в каком-нибудь другом месте. Этот подход полезен в том случае, когда функцию собираются вызывать несколько раз и из разных мест. Функция немедленного вызова — это отличный инструмент для описания и выполнения любого неповторяющегося задания.

Определение функции немедленного вызова берется в скобки вместе со списком параметров, который помещается в конце:

```
var result = (function() {
   ...
}());
```

Выражение, в котором находится функция немедленного вызова, говорит о том, что возвращается значение, а не сама функция. Если функции нужны параметры, то показанный ранее код можно переписать следующим образом:

```
var result = (function(x, y, z) {
   ...
}(1, 2, 3));  // 1,2,3 являются фактическими значениями для x,y,z
```

Точно так же, как и функции, можно определять *объекты немедленного вызова* (immediate objects). Объект немедленного вызова — это объект, определенный в том же самом месте, где вызывается и выполняется один из методов объекта. Определение объекта немедленного вызова заключается в скобки:

```
({
   init: function() {
     // решение задач инициализации
   },
}).init(...);
```

Объекты и функции немедленного вызова создают «песочницу» областей видимости, которая не дает их локальным переменным засорять глобальное пространство имен.

Что лучше применять, объекты или функции немедленного вызова? По большей части это зависит от вас, но в конечном счете все определяется сложностью вашего кода (или того, во что вы собираетесь его превратить). Для очень сложных задач предпочтительнее, наверное, объекты немедленного вызова, поскольку они позволяют определять свойства и разбивать реализацию на несколько методов. А функции немедленного вызова подходят для чего-нибудь попроще, поскольку удобны для выполнения любого кода, выраженного в виде простой последовательности инструкций.

Расширение существующих объектов путем добавления им поведения

Ранее рассматривался вопрос добавления свойств к прототипу объекта, чтобы каждый новый экземпляр объекта пополнялся этими новыми свойствами. Примерно те же соображения могут быть распространены на функции, которые могут использоваться для наделения существующего объекта поведением. В качестве примера рассмотрим еще раз объект `Number`. Порядок добавления нового элемента `random`, возвращающего случайное число больше указанного минимума, показан в следующем коде:

```
// Этот код нужно запустить один раз, поэтому он добавлен
 // в виде функции немедленного вызова.
 (if (typeof Number.prototype.random === 'undefined') {
   Number.prototype.random = function(min) {
     var n = min + Math.floor(Math.random() * 1000);
     return n;
   };
}());
```

После однократного выполнения этого кода вы сможете воспользоваться следующей функцией:

```
function doWork() {
   var n1 = new Number().random(0);
   var n2 = new Number().random(10);
   alert("Numbers are " + n1 + " and " + n2);
}
```

ПРИМЕЧАНИЕ

Добавление элементов в прототип исходных объектов считается дурным тоном, поскольку делает код менее предсказуемым и может затруднить его сопровождение. Однако эти рассуждения большей частью касаются командной разработки. Если код создается для себя, то этот пункт становится не столь важным.

Конструкторы

Как уже упоминалось, функции используются по двум основным причинам: для определения повторяющегося поведения и для создания нестандартных объектов. Разберем теперь вторую причину, для чего сначала рассмотрим следующий код:

```
var Dog = function(name) {
   this.name = name;
   this.bark = function() {
     return "bau";
   };
};
```

Здесь мы получаем новый объект по имени `Dog`. Ранее, когда речь шла об объектах, вы уже создавали нечто подобное:

```
var dog = {
   name: "Jerry Lee Esposito",
};
```

Существенная разница состоит в том, что последний фрагмент кода представляет собой фиксацию одних только данных, в то время как предыдущий фрагмент кода является функцией с данными и поведением.

Итак, как же использовать объект `Dog`? Для этого нужно, как показано далее, получить его экземпляр с помощью классического конструктора `new`:

```
var jerry = new Dog("jerry");
```

А что получится, если опустить оператор `new`:

```
var jerry = Dog("jerry");
```

Самое забавное, что, забыв указать оператор `new`, вы даже не получите никакого исключения, ваш код будет работать. Однако любое действие с объектом `Dog` (например, установка имени свойства) будет происходить с объектом `this`. Без оператора `new` это будет расценено как обращение к глобальному JavaScript-объекту. То есть этим вы засорите глобальное пространство имен JavaScript-интерпретатора. А вот как выглядит безопасная контрмера, не создающая проблем независимо от того, указан оператор `new` или нет:

```
var Dog = function(name) {
   var that = {};
   that.name = name;
   that.bark = function() {
     return "bau";
   };
   return that;
};
```

Разница в том, что теперь новый объект создается и возвращается явным образом — как объект по имени `that`. Обычно этот паттерн называется Use-That-Not-This (используйте that вместо this).

ВНИМАНИЕ

Это также наиболее приемлемый способ создания собственных агрегатов данных и поведения (объектов), в котором явно используется конструктор, а в операторе new нет необходимости. Таким образом, вы получаете в JavaScript очень близкое приближение к объектно-ориентированным компонентам.

Увлечение обратными вызовами

Функциями *обратного вызова* (callbacks) называются функции, передаваемые в качестве аргумента. Код, получающий обратный вызов, может обратиться снова к вызывающему коду, когда это необходимо. Предположим, к примеру, что у вас есть функция общего назначения, содержащая коллекцию чисел. В разное время вам нужно осуществить циклический перебор коллекции и выполнить различные операции, такие как сложение и умножение. Нужно ли определять две разные, но во многом повторяющие друг друга функции, как показано в следующем примере?

```
function sumAllNumbers() {
   // ШАГ 1: СБОР ВХОДНЫХ ДАННЫХ
   var numbers = new Array();
   for (var i = 0; i < arguments.length; i++) {
     numbers.push(arguments[i]);
   }

   // ШАГ 2: ВЫПОЛНЕНИЕ ОПЕРАЦИИ
   var result = 0;
   for (var i = 0; i < numbers.length; i++) {
     result += numbers[i];
   }

   // ШАГ 3: ВЫВОД РЕЗУЛЬТАТОВ
   alert("SUM result is: " + result);
 }

 function multiplyAllNumbers() {
```

продолжение ⇩

```
    // ШАГ 1: СБОР ВХОДНЫХ ДАННЫХ
    for (var i = 0; i < arguments.length; i++)
      numbers.push(arguments[i]);

    // ШАГ 2: ВЫПОЛНЕНИЕ ОПЕРАЦИИ
    var result = 1;
    for (var i = 0; i < numbers.length; i++) {
      result *= numbers[i];
    }

    // ШАГ 3: ВЫВОД РЕЗУЛЬТАТОВ
    alert("MULTIPLICATION result is: " + result);
}
```

Обе функции выполняются за три основных этапа. Первый шаг повторяется в обеих функциях, второй шаг свой для каждой из них, а третий шаг делает в обоих случаях то же самое, но с разными данными. Функции, определенные таким образом, конечно, работают. Если вас интересует только получение результата, то на этом можно остановиться.

Дело в том, что такие упрощенные решения, возможно, хороши для простых сценариев, но по мере усложнения программы вы можете столкнуться с большим объемом дублирующегося кода, а это плохо, поскольку вам придется вносить изменения сразу в нескольких местах, а если вы не внесете изменение в одном месте, могут возникнуть ошибки, которые трудно будет найти и исправить. Функции обратного вызова помогают не только тем, что сокращают объем кода, но и тем, что его становится проще читать.

Чтобы это доказать, начнем с создания единой функции, выполняющей операции, общие как для сложения, так и для умножения. Эту функцию можно назвать `handleNumbers`:

```
function handleNumbers(operationCallback) {
    // ШАГ 1: СБОР ВХОДНЫХ ДАННЫХ
    var numbers = new Array();
    for (var i = 1; i < arguments.length; i++) // начинаем с 1 для пропуска
                                               // функции обратного вызова
      numbers.push(arguments[i]);

    // ШАГ 2: ВЫПОЛНЕНИЕ ОПЕРАЦИИ
    var result = operationCallback(numbers);

    // ШАГ 3: ВЫВОД РЕЗУЛЬТАТОВ
    alert(result);
}
```

Теперь определим пару узкоспециализированных функций: одну для сложения, другую для умножения всех полученных параметров:

```
function doSum(numbers) {
    var result = 0;
    for (var i = 0; i < numbers.length; i++) {
```

```
    result += numbers[i];
  }
  return result;
}

function doMultiply(numbers) {
  var result = 1;
  for (var i = 0; i < numbers.length; i++) {
    result *= numbers[i];
  }
  return result;
}
```

На данный момент код вызова этой пары функций имеет следующий вид:

```
handleNumbers(doSum, 1, 2, 3, 4);
handleNumbers(doMultiply, 1, 2, 3, 4);
```

При такой структуре добавление еще одной операции над массивом чисел становится вопросом создания функции, выполняющей нужную операцию. Теперь уже не приходится волноваться о сборе чисел или выводе результатов.

Функции обратного вызова на основе контрактов

При тщательном сравнении фактических результатов этих двух версий кода можно заметить разницу. В первой версии с повторяющимся кодом можно вывести вполне понятное сообщение вроде «Результат сложения XXX». В более универсальном решении, связанном с применением функции обратного вызова, пользователю можно показать только голый числовой результат.

Звучит неприятно, но на самом деле все можно исправить.

Проблема в том, что в выводимое сообщение нужно включить информацию, касающуюся названия операции, а это может быть сделано только встроенной функцией. А как заставить функцию вроде `doSum` возвращать два значения: фактический результат операции и ее название? Вам нужно определить контракт для функции обратного вызова и фактически обновить ее, переведя из разряда простой функции в более высокий разряд объекта.

Сначала нужно определить контракт (или интерфейс), который, по вашему мнению, нужен функции обратного вызова. В контракте указывается вся информация, в которой нуждается вызывающий код. В данном случае вызывающий код, наверное, ожидает найти метод для вычисления числа (назовем его `execute`) и строку, указывающую на название операции (назовем ее `name`). Тогда функцию `handleNumbers` нужно переписать следующим образом:

```
function handleNumbers(operation) {
  // ШАГ 1: СБОР ВХОДНЫХ ДАННЫХ
  var numbers = new Array();
  for (var i = 1; i < arguments.length; i++)
```

продолжение ⇩

```
      numbers.push(arguments[i]);

   // ШАГ 2: ВЫПОЛНЕНИЕ ОПЕРАЦИИ
   var result = operation.execute(numbers);

   // ШАГ 3: ВЫВОД РЕЗУЛЬТАТОВ
   alert(operation.name + " result is " + result);
}
```

Теперь нужно определить два разных объекта, `Sum` и `Multiplication`:

```
var Sum = function () {
   var that = {};
   that.name = "SUM";
   that.execute = function (numbers) {
     var result = 0;
     for (var i = 0; i < numbers.length; i++) {
        result += numbers[i];
     }
     return result;
   };
   return that;
 }

 var Multiplication = function () {
   var that = {};
   that.name = "MULTIPLICATION";
   that.execute = function (numbers) {
     var result = 1;
       for (var i = 0; i < numbers.length; i++) {
         result *= numbers[i];
       }
     return result;
   };
   return that;
}
```

И наконец, нужен код для вызова операций:

```
handleNumbers(new Sum(), 1, 2, 3, 4);
handleNumbers(new Multiplication(), 1, 2, 3, 4);
```

Использование объектов вместо функций дает при программировании значительно больше возможностей, поскольку не ограничивает ни в чем в понятиях контракта, который можно поддержать.

Анонимные функции

До сих пор речь в основном шла об именованных функциях. А как насчет анонимных (безымянных) функций? Анонимные функции в JavaScript являются основой функционального программирования. Анонимная функция — это прямое ответвление лямбда-исчисления или, если хотите, языковая адаптация старомодных указателей на функции. Простой пример анонимной функции:

```
function(x, y) {
   return x + y;
}
```

Единственное отличие обычной функции от анонимной состоит в наличии (или отсутствии) имени.

Почему анонимные функции стали такими популярными? Главная причина в том, что анонимные функции позволяют определить нужный код прямо на месте, что исключает необходимость определять именованную функцию где-то еще. Недостаток анонимных функций заключается в том, что их нельзя использовать многократно. Если нужно всего лишь передать в качестве аргумента одноразовый фрагмент кода, то анонимные функции подходят для этого как нельзя лучше. Но если функция может потребоваться более одного раза, предпочтение следует отдать именованной функции. В качестве иллюстрации рассмотрим переделанный вариант предыдущего кода:

```
// Выполнение сложения
 handleNumbers(function(numbers) {
   var result = 0;
   for (var i = 0; i < numbers.length; i++) {
      result += numbers[i];
   }
   return result;
 },
1, 2, 3, 4);
```

Читается подобный код не всегда идеально, но для решения элементарных задач такой подход вполне приемлем.

Организация своего JavaScript-кода

До сих пор основное внимание уделялось синтаксическим аспектам языка JavaScript и изучению его объектно-ориентированных возможностей. Теперь настало время перейти к вопросу использования JavaScript-кода на HTML-страницах, а конкретно — в контексте приложений для Windows 8.

Любой используемый вами на HTML-станице JavaScript-код всегда вызывается в ответ на событие, которое инициируется либо браузером, либо каким-нибудь действием пользователя. Стало быть, первое, на что следует обратить внимание, — определение обработчиков событий на HTML-странице. Затем мы немного поговорим о том, как организовать код.

Привязка JavaScript-кода к страницам

В первую очередь нужно сказать, что *обработчики событий* являются JavaScript-функциями, вызываемыми в ответ на события. Событие может инициироваться

браузером, например, когда страница полностью загружена, или быть ответом на действие человека, например, когда пользователь щелкает кнопкой мыши. Для производства каких-либо видимых эффектов обработчики событий должны быть связаны с событиями. В HTML определяется ряд атрибутов вида `on`*XXX* (где *XXX* — имя события, например `click` или `load`), которые можно включить в HTML-разметку, чтобы связать событие с именем JavaScript-функции.

Ненавязчивый JavaScript-код (unobtrusive JavaScript) — это паттерн, предлагающий более эффективный способ получения такого же результата. Ненавязчивый JavaScript-код является также предпочтительным средством привязки кода к событиям в Windows 8. Давайте проясним ситуацию.

Атрибут event HTML-элемента

С годами вошло в привычку создавать HTML-страницы с полями ввода и кнопками, привязанными к JavaScript-обработчикам событий явным образом. Обычно это выглядит так:

```
<input type="button" value="Click me" onclick="handleClick()" />
```

С точки зрения чистой функциональности в этом коде все правильно — он работает, как и ожидалось: атрибут `onclick` связывает действие щелчка с JavaScript-функцией `handleClick`, и эта функция вызывается при щелчке пользователя кнопкой мыши. Однако такой подход приемлем главным образом только при использовании JavaScript в качестве «приправы» к простым HTML-страницам. Когда JavaScript-код составляет существенную долю страницы или представления, вся конструкция становится слишком громоздкой.

Ненавязчивый JavaScript-код является в известном смысле сценарным партнером CSS-классов. Используя CSS, вы пишете обычный HTML-код, не включая в него встроенной стилевой информации. Затем дизайнеры стилизуют элементы с помощью CSS-классы. Аналогично, в случае ненавязчивого JavaScript-кода вы избегаете использования атрибутов обработчиков событий, встраиваемых в теги (`onclick`, `onchange`, `onblur` и тому подобные), а вместо них, когда страница готова к визуализации, задействуете отдельную JavaScript-функцию для привязывания обработчиков. За счет того, что атрибуты событий не используются, код разметки и JavaScript-код отделяются друг от друга.

А как насчет кода, на который ссылаются обработчики событий? Иными словами, где должен быть код для таких функций, как `handleClick`?

Внедренный JavaScript-код

JavaScript-контент можно внедрить в HTML-страницу несколькими способами. Проще всего иметь готовый к использованию JavaScript-код на самой HTML-странице, поместив его внутри элемента `script`:

```
<script type="text/javascript">
    ...
</script>
```

Как только браузер находит один из таких разделов `script`, он прекращает вывод страницы, выполняет сценарий, а затем продолжает вывод. Если элемент `script` не содержит непосредственного кода для выполнения (предположим, к примеру, что он состоит лишь из определений функций), браузер просто берет функции на заметку и продолжает вывод страницы. Следовательно, элемент `script` является вполне допустимым местом внедрения определений функций, вызываемых обработчиками событий.

Внедрение кода сценария в страницу имеет свои положительные и отрицательные стороны, но отрицательных больше (табл. 4.3).

Таблица 4.3. Аргументы «за» и «против» внедрения JavaScript-кода в HTML-страницы

За	Против
Страница не зависит ни от чего другого и является самодостаточной. Разработчику не приходится заглядывать сразу в несколько мест, чтобы понять, в чем специфика страницы	Страница усложняется и требует больше времени на загрузку
	Код сценария не может использоваться за пределами содержащей его страницы
	Реструктуризация кода с целью минимизации дублирующейся функциональности осложняется отсутствием возможности его многократного использования вне страницы
	Код сценария не может быть кэширован браузером отдельно от страницы

Кроме того, есть еще одно обстоятельство, требующее рассмотрения: при наличии внедренного кода любые изменения сценария тут же становятся видимыми. Нужно лишь сохранить файл и обновить страницу в браузере. А когда файл сценария привязан в качестве внешнего ресурса (о чем мы вскоре поговорим), вам придется предпринять дополнительные действия, чтобы гарантировать немедленное внесение изменений, маскируемых кэшированными копиями того же самого файла. Этот вопрос чрезвычайно важен, но касается он не готовых приложений, а только фазы разработки.

Внешние файлы

Альтернативой внедренному в элемент `script` коду служит использование того же самого элемента `script` для ссылки на JavaScript-файл как на внешний ресурс. Вот как это выглядит:

```
<script src="http://..." type="text/javascript" />
```

Аргументы в пользу использования внешнего связанного JavaScript-файла прямо противоположны аргументам против внедрения сценариев. Именно этим путем и нужно идти, или, по крайней мере, рассматривать его в первую очередь как при веб-разработке в целом, так и при разработке приложений для Windows 8.

JQUERY И WINDOWS 8

В настоящее время JavaScript-разработка ведется в основном с помощью весьма популярной библиотеки jQuery, которую можно загрузить с сайта http://jquery.com.

Веб-разработчики используют библиотеку jQuery для ненавязчивой привязки обработчиков к событиям и выяснения факта наступления события на уровне страницы, например, события, свидетельствующего о готовности страницы к выводу на экран. Кроме того, веб-разработчики используют jQuery для выполнения запросов специализированных поднаборов элементов страницы. Вспоминая предыдущую главу, можно сказать, что jQuery предлагает синтаксис, напоминающий синтаксис CSS-селекторов. Однако в действительности синтаксис jQuery-селекторов намного богаче стандартов CSS.

Для разработчиков, имеющих богатый опыт веб-разработки, сама мысль написания HTML-страниц без использования jQuery (или других популярных JavaScript-библиотек, например knockout.js) может показаться абсурдной.

Применение jQuery (равно как и других библиотек, исключая WinJs) в Windows 8 может иногда порождать проблемы, зависящие от того, как именно используется библиотека jQuery.

Причина возникновения проблем заключается в новой модели безопасности, принятой Microsoft для приложений Windows 8. Согласно этой модели, динамическая манипуляция структурой страницы, производимая посредством потенциально небезопасных данных (то есть данных, полученных из ненадежного источника), запрещена.

Если вы хорошо знакомы с jQuery, то можете использовать эту библиотеку, если только это не создаст вам дополнительные сложности. Если вы не знакомы с jQuery, то я рекомендую не прибегать к этой библиотеке при разработке приложений для Windows 8. Для большинства нетривиальных функциональных возможностей библиотеки jQuery (например, для плагинов) вы найдете «родные» компоненты в библиотеке WinJs для Windows 8. Для запроса элементов главной HTML-станицы можно воспользоваться стандартными HTML-функциями, (например, document.querySelector), которые поддерживают синтаксис выбора элементов, похожий на CSS-синтаксис.

Привычки и наклонности

JavaScript-код можно легко заставить работать, но им весьма сложно управлять и его сложно развивать, пока не будет выработан и принят к исполнению ряд практических

приемов. Единственной целью данного раздела является обобщение всего того, что можно и что нельзя делать при простом JavaScript-программировании. Под «простым» JavaScript-программированием подразумеваются технические приемы, не направленные конкретно на разработку приложений для Windows 8. Специфические аспекты JavaScript-разработки для Windows 8 рассматриваются в следующей главе.

Группировка своих глобальных данных

В данной главе вы узнали о необходимости выделения своих глобальных данных. Это можно сделать, включив следующий код в верхнюю часть каждого JavaScript-файла:

```
var Globals = (function() { return this; }());
```

Предположим, что у вас есть глобальная переменная, представляющая имя приложения, пусть, к примеру, эта переменная называется `AppName`. Цель предыдущей инструкции состоит в том, чтобы эту переменную можно было использовать следующим образом:

```
Globals.AppName = "MyApp";
```

Следует отметить, что переменные `AppName` и `Globals.AppName` указывают на одно и то же место, и обе они могут использоваться в приложении с одинаковым смыслом. Разумеется, если постоянно задействовать версию с префиксом `Globals`, то ваши глобальные переменные будут прекрасно выделяться на фоне остального кода.

Но и этот подход не лишен проблем. В частности, в этом случае в глобальном пространстве имен создается столько записей, сколько переменных связано с `Globals`. Немного лучший подход выглядит так: сначала в верхнюю часть каждого, определяемого вами JavaScript-файла помещается следующий код:

```
Globals = Globals || {};
```

Далее глобальные элементы определяются как элементы только что созданного объекта `Globals`. Разница в том, что теперь `Globals` является совершенно новым глобальным объектом и все элементы, которые вы хотите использовать в своем приложении в качестве глобальных, теперь определены в виде дочерних членов объекта `Globals` и не засоряют пространство имен. Иначе говоря, все глобальные переменные вашего приложения группируются под одним глобальным объектом, который виден JavaScript-интерпретатору.

Сделайте доступным состояние приложения

Все приложения должны поддерживать свое состояние таким образом, чтобы оно могло быть сохранено на диске при закрытии приложения, а при использовании на мобильном устройстве переходило в фоновый режим.

Лучше создавать такой код, который будет готов к загрузке состояния приложения при запуске самого приложения. Данные могут быть загружены из места их хранения (если это приемлемо), из удаленного источника или просто инициализированы на основе исходных значений. При выходе из приложения текущее состояние должно быть сохранено в хранилище данных в готовности к дальнейшему использованию.

Для слаженной работы данной схемы необходимо начать с определения JavaScript-объекта, свойства которого представляют состояние приложения. Например:

```
var MyAppState = function () {
    var that = {};
    that.init = function () {
      // Обеспечить инициализацию и значения, предлагаемые по умолчанию
    };
    that.load = function () {
      // Обеспечить загрузку из хранилища
    };
    that.save = function () {
      // Обеспечить хранилище
    };

    // Сюда помещаются другие свойства
    ...

    that.init();
    return that;
}
```

Для прикрепления экземпляра этого объекта к контейнеру `Globals` нужен следующий код:

```
Globals.Current = new MyAppState();
```

Теперь можно считывать и записывать состояние вашего приложения посредством понятных инструкций, таких как `Globals.Current.`*Xxx*, где *Xxx* является именем свойства или члена, определенным в `MyAppState`.

Будьте готовы к локализации

Наличие приложения, готового к применению на международном рынке, может стать ключом к вашему успеху (или просто потешить ваше самолюбие, если вы подходите к разработке для Windows 8 без особых амбиций). В общем, локализация является важным фактором для приложений, публикуемых в общедоступном магазине, как в случае с приложениями для Windows 8.

В JavaScript-коде для веб-приложений вы не получите обширную встроенную справку по вопросам локализации. К счастью, при разработке приложений для Windows 8 вы получаете существенную поддержку от библиотеки WinJs. В конце дня вам понадобится лишь пометить специальным атрибутом те элементы, контент которых требуется локализовать. Затем вам нужно будет лишь добавить

к приложению файл локализованных ресурсов для каждого языка, который вы собираетесь поддерживать.

Автоматический выбор нужного контента из нужного файла ресурсов в соответствии текущими параметрами места пребывания возьмет на себя система времени выполнения Windows 8.

ПРИМЕЧАНИЕ

Если вы знакомы с C# и общей разработкой для Windows, то в этой общей схеме для вас не будет ничего нового. Это именно та старая добрая схема локализации Windows, которая адаптирована для нового прикладного программного интерфейса Windows 8.

Выводы

В данной главе предложен краткий обзор языка JavaScript, причем повествование шло без явного уклона в сторону Windows 8 или веб-технологий (по мере возможностей). Были рассмотрены основы языка и выделен ряд разумных схем эффективной разработки.

Честно говоря, эта глава, как и две предыдущие, касающиеся HTML и CSS, может послужить только для того, чтобы освежить или, возможно, уточнить знание соответствующих тем. Каждая из технологий, HTML, CSS и JavaScript, достойна отдельной книги для полного объяснения и поэтапного изучения. На случай, если у вас возникнет потребность в более конкретных источниках, обратитесь к следующим книгам издательства Microsoft Press.

Для новичков:

- «Start Here!™ Learn HTML5», Faithe Wempen (Microsoft Press, 2012);
- «Start Here!™ Learn JavaScript», Steve Suehring (Microsoft Press, 2012).

Для более искушенных читателей или для тех, кто хочет углубить свои знание после прочтения предыдущих книг:

- «HTML5 Step by Step», Faithe Wempen (Microsoft Press, 2011);
- «JavaScript Step by Step», Steve Suehring (Microsoft Press, 2013).

Стоит заметить, что в данной главе речь главным образом шла просто о языке JavaScript. Но JavaScript, по большому счету, является языком, используемым в HTML-страницах, а HTML-страницы, по сути, являются сетевыми ресурсами.

Со следующей главы все изменится. Вы увидите, что JavaScript-программирование для Windows 8 по своей природе несколько отличается от JavaScript-программирования для веб-приложений. Приложение для Windows 8 является, главным образом, приложением, ориентированным на прикладной программный

интерфейс Windows 8. A JavaScript является лишь средством, с помощью которого выражается логика и организуются вызовы к этому нижележащему прикладному программному интерфейсу.

В этом плане HTML и CSS — всего лишь языки пользовательского интерфейса, посредством которых этот интерфейс разрабатывается и выражается. Вы больше будете сосредоточены не на HTML5-элементах и собственных нестандартных CSS-таблицах, а на HTML-атрибутах Windows 8 и таблицах стилей. Все изученное в этих ранних главах пригодится в последующих. Но в HTML, CSS и JavaScript есть еще многое, о чем вам нужно знать и о чем в данной книге еще не рассказывалось.

5 Первые шаги в разработке приложений для Windows 8

Успех для тех заманчив, кто не пресыщен им.

Эмили Дикинсон

В главе 1 мы впервые кратко познакомились со стилем программирования для Windows 8. Мы создали первое простое приложение, воспользовавшись одним из основных шаблонов, предложенных Microsoft Visual Studio, превратив его затем в чуть более функциональное и сложное приложение, способное по вашему запросу показывать число, сгенерированное случайным образом. Как гласит народная мудрость, каждое, даже самое длинное, путешествие начинается с небольшого шага.

Чтобы выйти за рамки простого получения и вывода на экран случайного числа, нужно взять некоторые HTML- и CSS-команды и привязать их к языку JavaScript. Эти команды помогут вам представить себе и создать графическую часть любого разрабатываемого приложения, а язык JavaScript поможет организовать код таким образом, чтобы исключить неожиданные результаты и облегчить превращение ваших идей в команды операционной системе.

Теперь вы готовы приобщиться к среде выполнения Windows 8 (Windows 8 Runtime, WinRT). Среда выполнения представляет собой коллекцию работающих под

управлением Windows 8 программ и компонентов, взаимодействующих с любыми приложениями и заставляющих их работать. Эта среда предоставляет службы и данные любым приложениям, но также требует, чтобы приложения придерживались ряда правил и ограничений.

В этой главе преследуются три основные цели:

- изучение среды выполнения Windows 8;
- обзор графических требований к приложениям для Windows 8;
- понимание основных этапов жизненного цикла любого приложения для Windows 8.

Для достижения этих целей мы создадим еще одно учебное приложение, использующее основную службу среды выполнения — службу привязки данных. Эта служба предлагает пользователям вашего приложения простой способ вывода данных на экран. В следующих главах нам довольно часто придется задействовать службу привязки данных.

Среда WinRT

Ситуация, когда после появления новой версии операционной системы новые типы приложений оказываются абсолютно несовместимы с предыдущими версиями той же операционной системы, складывается достаточно редко. Иногда более новая версия операционной системы предлагает ряд новых функций, заставляющих компании обновлять существующие программы. Однако в Windows 8 имеется совершенно новый сегмент приложений, называемый приложениями Магазина Windows. Мало того, что эти приложения работают только под управлением Windows 8, они еще и выглядят совершенно по-другому и могут предлагаться бесплатно или продаваться в Магазине Windows, ну а последнее, хотя далеко не менее значимое, заключается в том, что их применение не ограничивается классическими персональными компьютерами и они могут запускаться без изменений на различных устройствах, совместимых с Windows 8, в первую очередь — на устройствах Microsoft Surface. Кроме того, приложения Магазина Windows могут разрабатываться только на машинах, работающих под управлением Windows 8.

ПРИМЕЧАНИЕ

Чтобы не дать разыграться чьим-нибудь преждевременным надеждам, следует заметить, что запустить ваше самодельное приложение для Windows на iPad не получится. Любые приложения для Windows 8 могут работать на устройствах только в том случае, если эти устройства совместимы с требованиями Windows к аппаратному и программному обеспечению.

Приложения Магазина Windows и все остальные приложения

В Windows 8 есть два рабочих режима, которые можно включать и выключать, — это стандартный режим Windows и новый режим Windows 8. При настройке под стандартный режим Windows ваша машина ничем особенным не отличается от старой доброй машины с операционной системой Windows 7. В этом режиме вы можете беспрепятственно запускать все имеющиеся у вас Windows-приложения — обратная совместимость полностью гарантируется. В конечном счете как пользователь вы не почувствуете большой разницы.

Однако при запуске машины под управлением Windows 8 она настраивается на работу в новом режиме. При этом вызывается внешне совершенно новый пользовательский интерфейс и становится доступен другой набор приложений. В этом режиме вы не найдете свои старые приложения, зато увидите все ваши новые самодельные приложения Магазина Windows (сказанное иллюстрирует рис. 5.1).

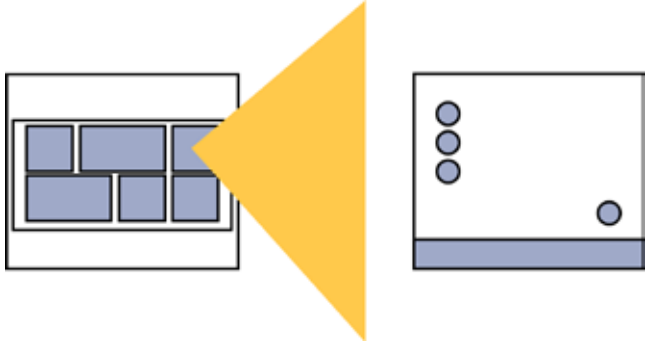

Рис. 5.1. Возвращение из Windows 8 к классическому пользовательскому интерфейсу Windows

ПРИМЕЧАНИЕ

Есть еще один взгляд на двоякую сущность Windows 8: классический интерфейс Рабочего стола Windows можно считать еще одним приложением, доступным в режиме Windows 8, если не считать того, что это встроенное приложение предлагает вам смотреть на вашу машину глазами операционной системы Windows 7.

Участь прежних Windows-приложений

Windows-приложения, ранее работавшие под управлением Windows 7, могут запускаться только в стандартном режиме Windows, поддерживаемом в Windows 8.

Чтобы иметь к ним доступ в исходном пользовательском интерфейсе Windows 8, их нужно переделать в приложения Магазина Windows. Например, без получения и установки версии Microsoft Office для Магазина Windows воспользоваться каким-либо из приложений Microsoft Office не удастся. То же касается браузеров: у обеих разновидностей Windows имеются собственные версии браузера Microsoft Internet Explorer.

Тем не менее в конечном счете Windows 8 не отказывается от совместимости с миллионами имеющихся приложений. Эта операционная система просто дает понять, что будущие платформы Windows последуют в другом направлении.

Поддерживаемые языки программирования

Приложения Магазина Windows можно создавать тремя основными способами. Можно использовать HTML и JavaScript, как показано в данной книге. Кроме того, приложения можно создавать с помощью языка программирования C# или Visual Basic и языка разметки XAML, предназначенного для определения пользовательского интерфейса. И наконец, можно задействовать язык C++ в сочетании с языком разметки XAML для задания пользовательского интерфейса.

С точки зрения эффективности программирования все три способа равно эффективны. Независимо от выбранных языка и средств разметки можно создавать одинаковые приложения. По ряду причин сочетание JavaScript и HTML дается легче большинству разработчиков, особенно начинающих (при сопоставимой эффективности).

ПРИМЕЧАНИЕ

Тот факт, что любой поддерживаемый платформой Windows 8 язык программирования может использоваться для создания приложений любого типа, не является каким-то побочным эффектом, а вытекает из общей архитектуры платформы Windows.

WinRT API

С выходом Windows 8 компания Microsoft сделала весьма важный шаг вперед. Именно он и стал причиной двойственности Windows 8 и привел к необходимости различать приложения Магазина Windows и другие приложения. В Microsoft Windows 8 заменен нижележащий (базовый) уровень, посредством которого ядро операционной системы функционирует и предоставляет свои возможности пользовательским приложениям.

Новая инфраструктура, положенная в основу приложений Магазина Windows, известна как Windows Runtime (WinRT).

Стек Windows 8

На рис. 5.2 представлен общий вид стека WinRT. На схеме показано, как два параллельных стека могут существовать бок о бок, поддерживая две различных модели разработки приложений, одна из которых основана на применении HTML и JavaScript, а другая — XAML и C# или Visual Basic. Следует заметить, что для простоты на рисунке не показан упомянутый ранее третий стек, основанный на применении C++ и XAML.

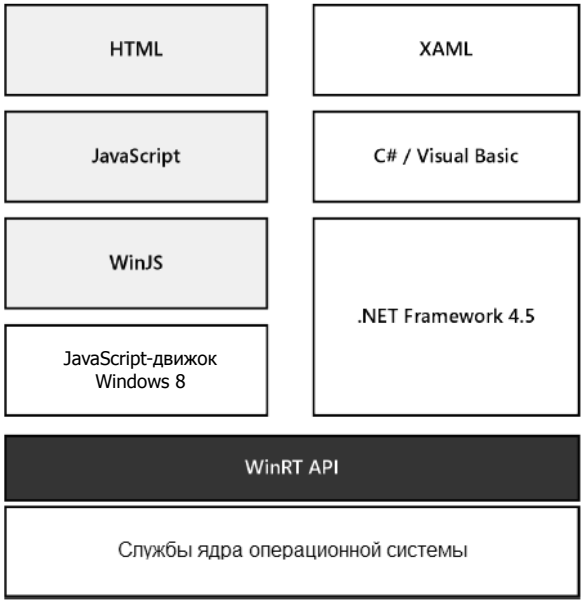

Рис. 5.2. Стек Windows 8

Важно отметить, что в основе обоих стеков лежат службы прикладного программного интерфейса (Application Programming Interface, API) среды WinRT, который, в свою очередь, обслуживается ядром операционной системы. Это ядро является основным механизмом, предоставляющим базовую функциональность. По аналогии с автомобилем, если WinRT API можно сравнить с двигателем, то ядро — с набором главных агрегатов этого двигателя.

Серым цветом выделены те блоки стека, о которых рассказывается в данной книге. Для стилизации пользовательского интерфейса мы используем HTML (и, возможно, CSS), а для придания ему желаемого поведения — JavaScript. Ваше взаимодействие с нижележащей операционной системой осуществляется через библиотеку WinJS. Любое приложение для Windows 8, написанное на JavaScript, включает в себя библиотеку WinJS и использует ее API для доступа к таким функциональным возможностям, как сохранение информации, работа в сети, вывод графики и т. д.

Любой написанный вами код будет компилироваться JavaScript-движком Windows 8, а затем выполняться. Доступ к подсистеме WinRT осуществляется в динамическом режиме при взаимодействии пользователя с приложением. На рис. 5.2 показано, что WinRT поддерживает все стеки: именно поэтому нет практически никакой разницы в функциональности приложений, написанных на JavaScript, и приложений, написанных на C#.

Возможности WinRT API

На рис. 5.3 развернут блок, который на рис. 5.2 назван WinRT API. Также на рис. 5.3 перечислены классы функций, доступные программистам, пишущим для Windows 8.

Рис. 5.3. Классы функциональности, доступные в WinRT

Для использования конкретного класса функциональности, например, для установки сетевого подключения и загрузки RSS-данных из удаленного URL-адреса, нужно просто сослаться на соответствующий JavaScript-файл в вашем приложении и приступить к использованию соответствующих функций.

Блок DirectX ссылается на нижележащий прикладной программный интерфейс, который поддерживает расширенные графические возможности верхних уровней стека Windows 8. Мультимедийный блок и блок представления предлагают инфраструктуру для обработки изображений и работы с ними, а также мультимедийные и визуальные элементы. Сетевой блок относится к коммуникационному уровню, главным образом — к HTTP-соединениям. И наконец, блок хранения относится к чтению и записи файлов и данных, а блок устройств предлагает набор функций для управления сенсорами и локально подключенными устройствами, например принтерами.

В следующих главах мы еще поговорим о различных классах функций и рассмотрим возможности их использования в учебных приложениях.

Аспекты применения WinRT API

Код, создаваемый в виде надстройки над WinRT API в рамках любого из поддерживаемых стеков разработки (в нашем случае HTML плюс JavaScript), будет иметь

ряд общих аспектов применения. Причиной тому являются конструкция и реализация нижележащего прикладного программного интерфейса WinRT, который в значительной степени определяет характер вызова функций через WinJS — ваш шлюз к этому низкоуровневому интерфейсу.

Например, любые функции, выполнение которых, как ожидается, должно длиться дольше нескольких миллисекунд, делаются асинхронными. Это существенное изменение для тех, кто уже имеет опыт разработки приложений для Windows. Асинхронными называются функции, начинающие свое выполнение сразу по вызову, но не имеющие четко обозначенного графика завершения своей работы.

То есть асинхронная функция сразу же после вызова продолжает выполняться параллельно с любым последующим кодом. А это означает, что разработчик, который создает код, использующий результаты асинхронной операции, столкнется с трудностями в поисках места размещения этого кода. В то же время переход к асинхронному программированию существенно смягчен рядом новых языковых возможностей. В конечном счете все, что вам нужно в качестве начинающего разработчика приложений для Windows 8, — это научиться вызывать функцию и получать результат ее работы. Однако для тех, кто сталкивался с .NET-разработкой, асинхронность в программировании, характерная для WinRT API, может стать проблемой.

Еще одна область применения WinRT API, заслуживающая предварительных размышлений, касается хранения данных и доступа к файлам. В приложениях Магазина Windows непосредственный доступ к файлам запрещен из соображений безопасности. Но это не означает, что вы не можете хранить данные в Магазине Windows на постоянной основе. Проще говоря, прикладной программный интерфейс хранения информации, к которому WinRT открывает доступ, не дает разработчикам полного контроля над диском, как в классических Windows-приложениях. Кроме того, для открытия и сохранения документов и файлов имеются новые компоненты, заменяющие стандартные и привычные для Windows диалоговые окна.

ПРИМЕЧАНИЕ

Разработчики приложений Магазина Windows располагают общей средой программирования (например, WinJS для тех, кто выбирает стек HTML плюс JavaScript), которая очень похожа на популярную среду .NET Framework, но все же имеет некоторые ключевые отличия. Эти отличия явно демонстрируют общую приверженность Microsoft к переработке и переосмыслению среды программирования, с которой имеют дело разработчики приложений для Windows 8. В результате некоторые классы в библиотеке WinJS функционально идентичны классам .NET Framework, у некоторых имеются незначительные отличия, у других классов вместо этого имеются аналоги, часть классов просто недоступна, и, наконец, появились некоторые новые классы. В целом все доступные классы вносят существенный вклад в этот мощный арсенал создания эффективных и функциональных приложений.

Пользовательский интерфейс приложений Магазина Windows

Если не брать в расчет функциональность, то принадлежность приложения к Магазину Windows определяется его пользовательским интерфейсом и возникающим у пользователя впечатлением от него. Этот новый революционный пользовательский интерфейс Windows 8 (Windows 8 UI) стал результатом сочетания функциональных возможностей, характерных для операционной системы Windows 8, с некоторыми инновационными дизайнерскими принципами.

Аспекты применения Windows 8 UI

Что касается внешнего вида, то принципы построения Windows 8 UI ориентированы исключительно на новые приложения Магазина Windows и не могут применяться ни к приложениям для Windows 7, ни веб-приложениям, ни к мобильным приложениям, даже если они разработаны и запущены под управлением Windows 8.

Если еще раз более пристально приглядеться к инициативе Windows 8 UI, вы не сможете не оценить тот ряд универсальных дизайнерских принципов, которые лаконично и довольно определенно описывают взгляд на прикладное программное обеспечение грядущих времен. Можно прийти к заключению, что в этой инициативе проглядывается суть нового восприятия программного обеспечения.

Первое представление о Windows 8 UI

Давайте кратко перечислим те аспекты применения, которые наиболее ярко характеризуют пользовательский интерфейс любого приложения Магазина Windows. Во-первых, следует отметить, что, хотя приложение Магазина Windows реагирует на прикосновения к экрану, оно не ограничено только лишь прикосновениями как единственным источником ввода. Поддержка прикосновений является тем аспектом, который позволяет вполне комфортно работать с приложениями на планшетных устройствах, а также на обычных ноутбуках и настольных компьютерах, оснащенных сенсорным экраном.

Во-вторых, приложения Магазина Windows выводятся на экран в едином окне, не имеющем границ и излишеств вроде перемещаемых границ, строки заголовка и значков. Это, несомненно, прорывные изменения по сравнению с предыдущими версиями Windows. В то же время приложения Магазина Windows адаптируются к действительным размеру и форме экрана и могут предложить по-настоящему гибкую визуализацию. Контент разумно адаптируется к той ориентации, которую имеет физический экран.

И наконец, приложение Магазина Windows хорошо интегрируется с окружающей средой и использует новые средства, такие как панель приложения (App bar),

Рис. 5.4. Выдвигающаяся панель Windows 8

Рис. 5.5. Примеры живых плиток Windows 8

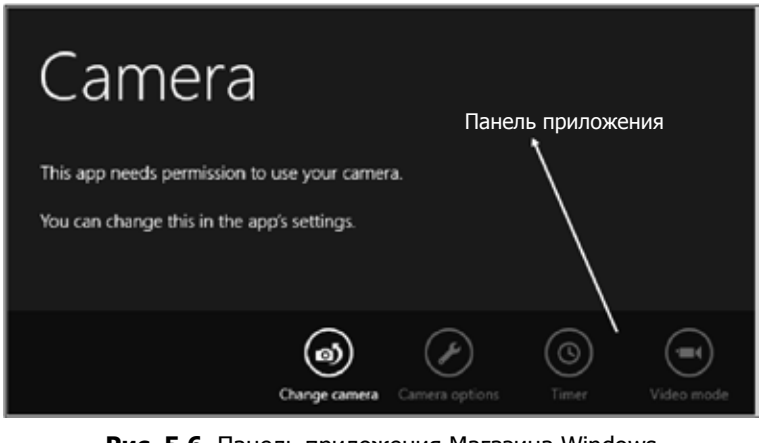

Рис. 5.6. Панель приложения Магазина Windows

живые плитки (Live tiles), пришедшие на смену прежним значкам рабочего стола, и выдвигающаяся панель (Charms bar) с кнопками быстрого доступа для

взаимодействия с другими приложениями посредством хорошо известных контрактов. В частности, живая плитка — это прямоугольная панель, выводимая на начальном экране Windows 8 и ссылающаяся на установленное приложение. «Живость» плитке придает возможность вывода небольшой анимации и контента на основе данных, содержащихся в приложении, или его текущего состояния. На рис. 5.4 показана выдвигающаяся панель, на рис. 5.5 — живые плитки, а на рис. 5.6 — панель приложения.

Дизайн, ориентированный на устройство

Приложение Магазина Windows представляется объединенным с дизайном операционной системы и полностью слитым с окружением. Вполне резонно считать, что применяемые принципы конструирования специфичны для платформы Windows 8. Точнее, это набор принципов, которые в плане разработки приложений для Windows 8 являются достаточно общими, чтобы стимулировать разработку любых программ и платформ, ориентированных на устройство.

Ориентированность на устройство (device-centric) — относительно новое понятие, которое, по сути, вращается вокруг идеи, что в основе дизайна и функциональности приложения должно лежать предположение о том, что приложение будет работать на множестве разнообразных устройств, отличающихся друг от друга размерами экрана, объемами памяти и программными возможностями.

Принципы создания Windows 8 UI

Считается, что принципов, вдохновивших на создание пользовательского интерфейса Windows 8, было семь:

- Вести разработку с ориентацией на прикосновения и интуитивно понятное взаимодействие.
- Оперативно реагировать на взаимодействие с пользователем и быть постоянно готовым к следующему взаимодействию.
- Снижать уровень избыточности вашего пользовательского интерфейса.
- Следовать существующей модели пользовательского интерфейса.
- Не изобретать колесо заново.
- Взаимодействовать с другими приложениями посредством контрактов.
- Учитывать облачные технологии.

Следование этим принципам действительно позволит сделать любые приложения, а не только приложения Магазина Windows, намного более эффективными и привлекательными.

ПРИМЕЧАНИЕ

Чтобы понять важность принципов дизайна Windows 8 UI, следует учесть, что в течение многих лет пользователям прикладных программ навязывались понятия и действия, соответствующие правилам организации дизайна приложения. Несмотря на лучшие побуждения, в программном обеспечении зачастую недостаточно проявлялась забота о конечных пользователях. Однако сейчас наметились перемены, появилось новое поколение пользователей, менее терпимое по сравнению с предыдущим поколением. Нужно быть готовым к разработке приложений, более гибких в отношении дизайна, причем как в графическом, так и в логическом аспекте. Возможно, вскоре такое программное обеспечение станет единственно востребованным. Благодаря естественному пользовательскому интерфейсу, вполне вероятно, что Windows 8 может реально упростить задачу создания такого программного обеспечения.

Естественно возникает вопрос: как превратить эти руководящие принципы в конкретные действия?

Дизайн прикосновений

Прикосновение — это то, что заставляет приложение быть готовым к немедленному использованию, что стало главным фактором, определившим успех смартфонов и планшетных устройств. Но этот же фактор является своеобразным водоразделом. Если вы разработали пользовательский интерфейс, рассчитанный на прикосновения, ваше программное обеспечение не сможет нормально работать на несенсорном устройстве. Следовательно, важно никогда не рассматривать прикосновение в качестве единственного источника ввода.

Это, к примеру, означает поддержку мыши или клавиатуры для классических пользователей и сенсорного экрана для молодых (или более привычных к нему) пользователей. Тогда пользователи смогут, скажем, сводить и разводить пальцы на сенсорном экране для масштабирования выводимой информации на планшетных устройствах и добиваться того же самого эффекта щелчком (или, возможно, двойным щелчком) кнопки мыши на ноутбуках.

Оперативная реакция и готовность

Прикосновения стали качественным скачком в отношении восприятия дизайна пользователем. Они имели сопутствующий эффект, связанный с еще одним аспектом восприятия пользователем — оперативной реакцией. Когда пользователь видит визуальные элементы и «трогает» их, он ожидает от них немедленной реакции, как и от всех других объектов реального мира. Попробуйте слегка стукнуть по какому-нибудь предмету, лежащему на краю стола. Если прикосновение возымеет действие, предмет тут же упадет: вам не придется в течение нескольких секунд созерцать значок песочных часов, ожидая, пока предмет упадет.

Кроме прикосновения и отклика на него важную роль во взаимодействии играет интуиция. Цель состоит в том, чтобы пользователям на любом этапе работы

с приложением было понятно, каковы должны быть следующие операции. Этот, несомненно, базовый принцип приводит к нетривиальным подходам в области организации дизайна пользовательского интерфейса, данных и логики.

Нулевая избыточность

Многие годы избыточность в пользовательском интерфейсе считалась достоинством. Избыточным пользовательский интерфейс считается, если он позволяет одну и ту же задачу решить более чем одним способом. Навигацию с помощью клавиатуры и мыши, слегка отличающиеся друг от друга пункты меню (например, Сохранить, Сохранить все, Сохранить как), клавиши быстрого вызова, приводящие к одному и тому же результату, сейчас можно встретить во многих приложениях.

В пользовательском интерфейсе, основанном на прикосновениях, существование всех этих методов уже не имеет смысла. Следовательно, политика нулевой избыточности является весьма ценным достоинством приложений Магазина Windows.

Такие же конструкторские приоритеты хорошо просматриваются во многих успешных мобильных приложениях, и они тесно связаны с возрастающими потребностями в плане простоты и эффективности, проявляющимися в возможности совершения большего количества действий (или просто одних и тех же действий) с меньшим количеством вариантов и меньшим количеством инструментов.

Поступайте как римляне

Прежде еще одним основным принципом разработки программного обеспечения, подвергнутым сомнению в Windows 8 UI, была всеобщая модель пользовательского интерфейса. По разным причинам в некоторых приложениях обязательно требовалось обеспечить возможность настройки пользовательского интерфейса. Иногда это делалось просто для того, чтобы выделиться в лучшую сторону среди приложений такого же типа. А иногда таким образом разработчики пытались сделать свои приложения более полезными и удобными.

Любое приложение в Windows 8 должно занимать собственную нишу в контексте модели пользовательского интерфейса, диктуемой платформой. Поэтому четвертый принцип разработки приложений для Windows 8 может быть сведен к старой поговорке, предлагающей вести себя подобно римлянам, по крайней мере, когда вы находитесь в Риме. К тому же этот принцип далеко не нов для тех, у кого имеется опыт в разработке мобильных приложений.

Приложения, особенно на рынке iOS и Windows Phone, могут отвергаться, если их пользовательский интерфейс идет вразрез с привычными нормами. То же относится к приложениям Магазина Windows.

Поступайте как римляне (а не изобретайте колесо заново)

Пятый принцип приложений Магазина Windows является слегка усложненным продолжением четвертого. Он еще больше подчеркивает важность наделения приложения таким пользовательским интерфейсом, который лучше соответствует системному пользовательскому интерфейсу. Речь идет не только о приведении приложения в соответствие с системой, но и о достижении этого за счет выбора тех инструментов и шаблонов, которые предлагаются стеком Windows 8.

Именно поэтому в следующих главах активно применяется упомянутая ранее библиотека WinJS. Эта библиотека является хранилищем (репозитарием) исходных инструментов, предназначенных для создания приложений Магазина Windows средствами HTML и JavaScript. С ней мы познакомимся в следующем разделе.

Взаимодействуйте с другими приложениями (а не изобретайте колесо заново)

Полная интеграция с используемой платформой является достоинством любого приложения Магазина Windows, поскольку дает пользователям возможность чувствовать себя в привычной обстановке при работе практически с любым приложением. Больше того, она позволяет различным приложениям взаимодействовать друг с другом и вести обмен данными.

Теперь можно сформулировать и шестой принцип, сказав, что приложение с целью реализации своей функциональности может опираться на доступные сервисы, предлагаемые другими приложениями. Приложения Магазина Windows могут импортировать функции из других приложений посредством контрактов. *Контракт* — это формализованный прикладной программный интерфейс, позволяющий приложениям вызывать функции других приложений. Тем самым вы освобождаетесь от необходимости многократно переписывать одни и те же функции, а пользователи получают возможность выполнять одно и то же действие практически одинаково.

Например, приложения Магазина Windows могут поддерживать системный контракт поиска (`Search`) для извлечения информации из других приложений и реализовывать контракт совместного использования (`Share`) для предоставления своего контента в общий доступ.

Над нами только облако

И наконец, седьмой принцип, наверное, не нуждается в объяснении. Локальные диски больше не являются единственным местом хранения и чтения данных. Чтобы поддерживать постоянный контакт пользователя с приложением, разработчики в течение многих лет использовали для сохранения личных данных cookie-файлы и локальные параметры. Облако просто добавляет еще одно измерение, публикуя личные данные и делая эту информацию доступной другим для социального взаимодействия программ. В данном случае облако рассматривается в качестве

специфичного для пользователя или даже для приложения хранилища, существующего на некоем удаленном общедоступном сервере.

Компоненты уровня представления

Библиотека WinJS состоит из JavaScript-объектов, целенаправленно разработанных, чтобы обеспечить удобный доступ к функциональным возможностям ядра Windows 8 и, следовательно, упростить разработку приложений Магазина Windows с помощью JavaScript.

WinJS состоит из двух основных частей: коллекции поведенческих объектов для решения задач уровня ядра, таких как хранение данных, работа в сети, работа с мультимедиа, поддержка жизненного цикла приложения, и набора виджетов для оформления пользовательского интерфейса. Далее в книге описываются обе части библиотеки WinJS. И в любом случае полезно будет получить общее представление о визуальных виджетах, которые можно быстро встраивать в ваши приложения в качестве строительных блоков.

Визуальные WinJS-элементы

В табл. 5.1 перечислены визуальные элементы библиотеки WinJS. Пользовательский интерфейс ваших будущих приложений Магазина Windows создается встраиванием одного или нескольких таких элементов в HTML-шаблон.

Таблица 5.1. WinJS-виджеты

Виджет	Описание
AppBar	Горизонтальная командная панель, которая обычно помещается в нижней части окна
DatePicker	Календарь, предоставляющий пользователю возможность выбрать дату
FlipView	Коллекция элементов, выводимая на экран по одному и позволяющая пользователю переключаться между ними. В качестве типичного примера можно привести вывод изображений и подписей с возможностью горизонтального пролистывания
Flyout	Простое всплывающее сообщение, исчезающее только при прикосновении к какому-нибудь месту экрана (или щелчке на нем кнопкой мыши)
ListView	Коллекция элементов, выводимая на экран в различных вариантах, например, в виде узлов сетки или в виде списка
HtmlControl	Любой контент, предоставленный в HTML-формате
Menu	Всплывающий объект, похожий на стандартное меню с перечнем команд

Виджет	Описание
PageControl	Агрегат из HTML-, JavaScript- и CSS-кода, который может быть внедрен в качестве представления или элемента навигации во внешнюю страницу. Элемент PageControl служит для определения этого агрегата
SemanticZoom	Два различных представления одного и того же контента, дающие пользователю возможность переключаться между ними. Один вариант является уменьшенным представлением контента, второй — увеличенным
TimePicker	Этот элемент дает пользователю возможность выбрать время в привлекательном графическом представлении
ToggleSwitch	Стандартный пользовательский интерфейс, дающий пользователю возможность включить или выключить некий режим
Tooltip	Всплывающее представление, которое может быть составлено из весьма сложного контента, например изображений и отформатированного текста. Назначение элемента Tooltip — предоставление дополнительной, но необязательной информации относительно того или иного элемента данных в представлении (всплывающая подсказка)
ViewBox	Элементы, выводимые внутри элементы ViewBox, автоматически масштабируются, заполняя доступное пространство. Кроме того, виджет ViewBox реагирует на изменения размера контейнера, например, после поворота экрана или изменения размера окна

Эти визуальные компоненты покрывают довольно большое количество сценариев, а от сторонних производителей, участников проектов с открытым кодом и публикаций в блогах можно ожидать дополнительных готовых к использованию визуальных компонентов Магазина Windows. По мере ознакомления с разработкой для Windows 8 вы даже можете приступить к созданию собственных виджетов. Наличие виджета, предлагающего ту или иную функциональность, способствует ее многократному использованию в разных страницах и приложениях.

ВНИМАНИЕ

При создании приложений Магазина Windows определение пользовательского интерфейса главным образом задается средствами HTML и CSS. При этом разрешается все, что можно сделать с помощью HTML и CSS. А тот ассортимент визуальных элементов, который представлен в табл. 5.1, просто помогает получить доступ к полезным компонентам и придать вашим приложениям согласованные вид и функциональность.

Предоставление пользовательского интерфейса по запросу

К данному моменту у вас должно уже сложиться представление о том, что Windows 8 предлагает пользователям несколько стандартных способов обращения к функциональным возможностям приложения. Есть статические команды, привязанные

к конкретным элементам пользовательского интерфейса (например, к кнопкам), и динамические команды, которые становятся доступными по запросу.

Используя принцип предоставления пользовательского интерфейса по запросу, вы делаете свое приложение максимально понятным и аккуратным, не перегружая его визуальными элементами. В то же время элементы, необходимые для запуска команд и начала операций, выводятся по запросу, когда в них нуждается пользователь. Панель приложения и выдвигающаяся панель являются именно такими механизмами, предоставляемыми системой Windows 8 по запросу для пользовательского интерфейса. Панель приложения является хранилищем команд приложения, которые становятся доступными, когда пользователь переходит к нижней части экрана. А выдвигающаяся панель появляется с правого края экрана.

Выдвигающаяся панель предлагает пользователям единообразный способ доступа к функциональным возможностям практически любого приложения. Это могут быть, например, поиск, совместное использование и доступ к файлам. Панель приложения, в свою очередь, содержит команды конкретного приложения, то есть считается, что доступные на ней команды имеют смысл только для текущего представления.

Создание учебной панели приложения

Чтобы лучше познакомится с программированием с использованием библиотеки WinJS, давайте проверим на практике, в чем заключается создание виджета `AppBar` в приложении Магазина Windows. В первую очередь, нам понадобится некая HTML-разметка, в которой будет содержаться виджет `AppBar`. Большинство визуальных компонентов библиотеки WinJS просто берут HTML-сегмент и превращают его во что-нибудь другое. Вы можете создать новый чистый проект и отредактировать страницу default.html, вставив в нее следующий код:

```
<!DOCTYPE html>
 <html>
 <head>
   <meta charset="utf-8" />
   <title>My AppBar</title>

   <!-- WinJS-ссылки -->
   <link href="//Microsoft.WinJS.1.0/css/ui-dark.css" rel="stylesheet" />
   <script src="//Microsoft.WinJS.1.0/js/base.js"></script>
   <script src="//Microsoft.WinJS.1.0/js/ui.js"></script>

   <!-- MyAppBar-ссылки -->
   <link href="/css/default.css" rel="stylesheet" />
   <script src="/js/default.js"></script>
 </head>
 <body>
   <h1 id="header">
     Sample page using an app-bar.
   </h1>
   <hr />
   <div id="yourAppBar" data-win-control="WinJS.UI.AppBar" data-win-options="">
```

```
      <button data-win-control="WinJS.UI.AppBarCommand"
              data-win-options="{id:'cmdAdd',label:'Add',icon:'add'}">
      </button>
      <button data-win-control="WinJS.UI.AppBarCommand"
              data-win-options="{id:'cmdRemove',label:'Remove',icon:'remove'}">
      </button>
      <hr data-win-control="WinJS.UI.AppBarCommand"
          data-win-options="{type:'separator',section:'global'}" />
      <button data-win-control="WinJS.UI.AppBarCommand"
              data-win-options="{id:'cmdDelete',label:'Delete',icon:'delete'}">
      </button>
      <button data-win-control="WinJS.UI.AppBarCommand"
              data-win-options="{id:'cmdCamera',label:'Camera',icon:'camera'}">
      </button>
   </div>
 </body>
</html>
```

Тело этой страницы содержит обычный элемент `div` и несколько элементов `button`. Таинство превращения этой разметки в виджет `AppBar` в стиле Windows 8 кроется в атрибуте `data-win-control`.

При использовании этого атрибута с тегом `div` он превращает его в объект `WinJS.UI.AppBar`, то есть в виджет `AppBar`. Точно так же при использовании его с элементом `button` он превращает его в кнопку на панели приложения или в объект типа `WinJS.UI.AppBarCommand`. Атрибут `data-win-options` содержит управляющие параметры. Например, кнопке виджета `AppBar` присваиваются уникальный идентификатор (`id`), подпись (`label`) и значок (`icon`). Для вывода виджета `AppBar` на экран нужно либо провести пальцем вверх, начиная от нижней части экрана, либо щелкнуть правой кнопкой мыши в окне приложения. Тот же результат можно получить, нажав сочетание клавиш Windows+Z.

А как насчет того действия, которое должно быть выполнено по щелчку пользователем на кнопке?

Только что введенная HTML-разметка служит лишь для определения макета панели. Чтобы добавить поведение, нужен еще JavaScript-код. Отредактируйте содержимое файла default.js следующим образом:

```
(function () {
   "use strict";

    WinJS.Binding.optimizeBindingReferences = true;

    var app = WinJS.Application;
    var activation = Windows.ApplicationModel.Activation;

    app.onactivated = function (args) {
       if (args.detail.kind === activation.ActivationKind.launch) {
          if (args.detail.previousExecutionState !==
              activation.ApplicationExecutionState.terminated) {
              // Загрузка состояния приложения, если это необходимо
         } else {
```

продолжение ⇩

```
            // Восстановление состояния приложения
        }
        args.setPromise(WinJS.UI.processAll()
            .then(init()));
    }
  };

  app.oncheckpoint = function (args) {
      // Это приложение будет приостановлено
  };

  app.start();
})();

// Функции кнопок
function doClickAdd() {
  var alertDialog = new Windows.UI.Popups.MessageDialog(
   "Add button clicked!");
  alertDialog.showAsync();
}

function init() {
  var page = WinJS.UI.Pages.define("default.html", {
    ready: function (element, options) {
      document.getElementById("cmdAdd").addEventListener(
        "click", doClickAdd, false);
    }
  });
}
```

Заданный как функция немедленного вызова, этот код определяет страницу и несколько обработчиков событий для этой страницы. Когда страница будет готова к взаимодействию с пользователем, может быть, к примеру, запущен код, связанный с событием `ready`. Его работа сводится к добавлению обработчика к событию щелчка `click` для каждой ранее определенной кнопки. Давайте детально рассмотрим код для события `click`:

```
document.getElementById("cmdAdd").addEventListener("click", doClickAdd);
```

В первую очередь код извлекает из страницы HTML-элемент, чей атрибут `id` имеет строковое значение `cmdAdd`. Этот HTML-элемент добавляется в качестве слушателя события `click`. Слушатель события представляет собой фрагмент JavaScript-кода, который запускается автоматически, как только пользователь инициирует конкретное событие, например щелкнет на кнопке или ткнет в нее. Тогда щелчок на кнопке, имеющей метку `Add`, запустит функцию `doClickAdd`. В частности, функция `doClickAdd` выводит сообщение, показанное на рис. 5.7.

ПРИМЕЧАНИЕ

Этот простой пример — всего лишь иллюстрация тех схем, которые применяются во всех остальных главах. По сути, каждая из будущих глав содержит перечень упражнений, позволяющих вам проверить на практике различные аспекты программирования для Windows 8 с использованием HTML и JavaScript.

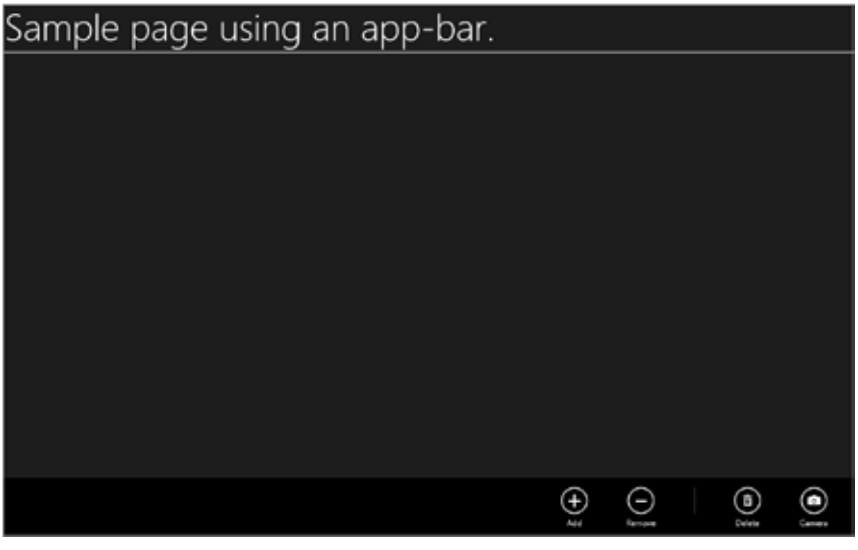

Рис. 5.7. Учебная панель приложения в действии

Привязка данных

В главе 1 мы создали свое первое приложение для Windows 8, генерирующее случайное число. В тот момент, когда число было сгенерировано, возник вопрос о его показе пользователю. Что касается решения этого вопроса, то, как было показано ранее, первым вариантом является использование WinJS-регистратора посредством метода `WinJS.log`. Но WinJS-регистратор хорош при тестировании, но не при работе реального приложения, поскольку он показывает выходную информацию только в Visual Studio.

Другим вариантом является использование окна сообщения, то есть всплывающего окна, показывающего текст и закрываемого пользователем вручную щелчком на одной из выводимых кнопок. Несмотря на то что такой вариант при некоторых условиях вполне приемлем, он для пользователя слишком навязчив. С его помощью хорошо задавать вопросы и в отдельных редких случаях демонстрировать какие-то данные. Но чаще всего нужно, чтобы приложение сгенерировало выходное значение и просто показало его через существующий пользовательский интерфейс. Иными словами, вам нужно привязать данные к существующему пользовательскому интерфейсу.

Программные манипуляции HTML-страницей

Приложение Магазина Windows, написанное с помощью HTML и JavaScript, в первую очередь является приложением, содержащим веб-страницы, обрабатываемые немного другим способом по сравнению с классическим браузером. С учетом этого приложение Магазина Windows может содержать любой JavaScript-код, в присутствии которого на веб-странице есть какой-нибудь смысл.

Когда браузер визуализирует веб-страницу, он создает в памяти представление ее контента, при этом каждый HTML-элемент превращается в программируемый объект. Это представление в памяти называется объектной моделью документа (Document Object Model, DOM).

Объекты из HTML-страницы извлекаются с использованием следующей строки кода:

```
var element = document.getElementById("идентификатор_элемента");
```

В качестве параметра методу `getElementById` передается уникальный идентификатор извлекаемого элемента. Более сложное выражение запроса может быть составлено с помощью синтаксиса CSS-селекторов, как показано в главе 3. В этом случае используемый код может выглядеть следующим образом:

```
// Запрос нескольких элементов
 var elements = document.querySelectorAll("ваше_css-выражение");

 // Запрос одного элемента с остановкой при первом совпадении
var element = document.querySelector("ваше_css-выражение");
```

Как только элемент для обновления обнаруживается, его контент можно изменить присваиванием свойству `innerHtml` любого строкового значения, в котором также может содержаться HTML-разметка. Показанный далее код извлекает элемент, названный `header`, и устанавливает для него строковое значение «`Hello`», которое визуализируется как полужирный текст:

```
var element = document.getElementById("header");
element.innerHtml = "<b>Hello</b>";
```

Эта схема демонстрирует программный способ обновления пользовательского интерфейса вычисленными данными. Подход, основанный на непосредственном обновлении DOM, обеспечивает очень быстрое выполнение кода, но подходит только для обновления конкретных элементов. Однако он неприменим в сценариях, где выводятся сразу несколько элементов, например в списках, или где несколько фрагментов данных выводятся на разных элементах.

ПРИМЕЧАНИЕ

Используя вместо свойства innerHtml свойство innerText, вы просто задаете для HTML-элемента обычный текст, не затрагивая разметку или стиль. В определенном смысле свойство innerText безопаснее, поскольку с этим свойством не возникает риска изменения существующей графической структуры.

Декларативные манипуляции HTML-страницей

Библиотека WinJS поддерживает также декларативную форму обновления HTML-элементов, известную как привязка данных. Давайте посмотрим, как она работает.

Создайте новый пустой проект приложения для Windows 8 и отредактируйте содержимое элемента body в файле default.html, придав ему следующий вид:

```
<!DOCTYPE html>
 <html>
 <head>
    <meta charset="utf-8" />
    <title>DataBinding</title>

   <!-- WinJS-ссылки -->
    <link href="//Microsoft.WinJS.1.0/css/ui-dark.css" rel="stylesheet" />
    <script src="//Microsoft.WinJS.1.0/js/base.js"></script>
    <script src="//Microsoft.WinJS.1.0/js/ui.js"></script>

    <!-- DataBinding-ссылки -->
    <link href="/css/default.css" rel="stylesheet" />
    <script src="/js/default.js"></script>
 </head>
 <body>
    <header>
       <h2>Start Here! Build <b>Windows 8</b> Applications with <b>HTML5</b> and
          <b>JavaScript</b></h2>
       <hr />
       <p>Random number displayed via data-binding.</p>
    </header>
    <div class="center">
       <span id="numberContainer" data-win-bind="innerText: generatedValue"></span>
    </div>
    <div class="center">
       <input id="numberButton" type="button" value="Get Number" />
    </div>
    <footer>
       <hr />
       Dino Esposito | Francesco Esposito
    </footer>
 </body>
</html>
```

Элемент span снабжен атрибутом data-win-bind. Являясь свойством, интерпретируемым средой выполнения Windows 8, этот атрибут производит магические действия по заданию контента элемента посредством свойства innerText. А что собой представляет строка generatedValue, следующая далее в этом фрагменте кода? Замысел состоит в генерировании случайного числа и выводе его через пользовательский интерфейс.

Хотя к HTML-элементу можно привязать данные любого типа, включая отдельные фрагменты данных, например строку или число, вам, скорее всего, понадобится проделывать это с составными объектами, получающимися из сочетания текста, чисел и данных. Давайте теперь предположим, что у вас есть JavaScript-объект, описывающий привязываемые данные. Добавьте в нижнюю часть файла default.js следующий сценарий:

```
// Получение объекта с внедренным случайным числом в диапазоне от 1 до 100
 function displayGeneratedNumber() {
```

продолжение ⇗

```
    var randomNumber = { generatedValue: Math.floor((Math.random()*100)+1) };

    // сюда помещается весь остальной код
}
```

Теперь у вас есть JavaScript-объект со свойством `generatedValue`, который содержит сгенерированное число, принадлежащее диапазону от 1 до 100. В соответствии с этим следующая разметка приобретает более важную роль:

```
<span id="numberContainer" data-win-bind="innerText: generatedValue"></span>
```

Строка `generatedValue` является выражением, ссылающимся на значение, которое вы собираетесь присвоить свойству `innerText`. Однако здесь нужно прояснить еще один аспект: как среда выполнения Windows 8 узнает об объекте, который представляет свойство `generatedValue`? Вам нужно вернуться к файлу default.js и немного переделать функцию `displayGeneratedNumber`:

```
// Получение объекта с внедренным случайным числом в диапазоне от 1 до 100
 function displayGeneratedNumber() {
    var randomNumber = { generatedValue: Math.floor((Math.random()*100)+1) };
    // Включение режима привязки выбранного HTML-элемента
    var bindableElement = document.getElementById("numberContainer");
    WinJS.Binding.processAll(bindableElement, randomNumber);
}
```

Добавленные две строки сообщают среде выполнения Windows 8, что элемент по имени `numberContainer` (это элемент `span`) должен быть обработан и привязан к любому контенту, который он может получить из объекта `randomNumber`. Теперь становится полностью понятной следующая разметка:

```
<span id="numberContainer" data-win-bind="innerText: generatedValue"></span>
```

Свойство `innerText` элемента `span` с именем `numberContainer` будет выводить значение, присвоенное свойству `generatedValue` привязанного объекта `randomNumber`. И это все? Нет, еще не все.

У нас пропущена ссылка там, где нужно запускать код, записанный в файле default.js. Придется еще раз отредактировать файл default.js. Изменить требуется строку, вызывающую `setPromise`:

```
app.onactivated = function (args) {
    if (args.detail.kind === activation.ActivationKind.launch) {
       if (args.detail.previousExecutionState !==
           activation.ApplicationExecutionState.terminated) {
             // Здесь происходит инициализация приложения.
             document.addEventListener(
               "DOMContentLoaded", displayGeneratedNumber);
       } else {
             // Здесь восстанавливается состояние приложения.
       }
       args.setPromise(WinJS.UI.processAll()
       .then(init()));
    }
};
```

Затем нужно добавить в нижнюю часть файла default.js новую нестандартную функцию под названием `init`:

```
function init() {
    document.getElementById("numberButton").addEventListener(
      "click", numberButtonClick);
}
```

Какие уроки можно извлечь из этого примера?

В первую очередь, привязка данных является мощным механизмом, раскрывающим всю свою эффективность в по-настоящему сложных ситуациях. Обращаться к декларативной привязке данных, если нужно вывести лишь один фрагмент данных, наверное, излишне. В таком случает больше подойдет непосредственное использование свойства `innerHtml`. А вот при наличии нескольких элементов или нескольких фрагментов одного элемента данных, привязка данных оказывается предпочтительнее, и ваша задача сводится к управлению неоднородными данными в виде объекта, добавлению атрибута к разметке и настройке нескольких строк кода для включения режима привязки.

ВНИМАНИЕ

Привязка данных в Windows 8 является одним из сервисов, которые предлагаются библиотекой WinJS. Для привязки данных использовать саму среду Windows 8 вам не придется. Это, возможно, самый простой выход, если у вас нет достаточных знаний в области веб-разработки. Но если вы уже знакомы с веб-разработкой и такими библиотеками, как Knockout (или с некоторыми дополнительными модулями jQuery), можете просто импортировать эти библиотеки и осуществить привязку данных через их инфраструктуру. Таким образом, в WinJS может работать любой известный вам подход. Если не получится, можно всегда воспользоваться великолепной инфраструктурой привязки данных, заложенной в саму библиотеку WinJS.

ПРИМЕЧАНИЕ

В данной главе вопрос привязки данных и довольно простой сценарий ее использования рассматриваются весьма поверхностно. В последующих главах рассказывается о привязке к пользовательскому интерфейсу сложных объектов и списков элементов посредством расширенной инфраструктуры WinJS.

Понятие жизненного цикла приложения

Любое приложение в Windows 8 характеризуется последовательностью событий, обозначающих запуск, загрузку и завершение работы приложения. Более подробное изучение этих событий, которые в совокупности составляют жизненный цикл приложения, имеет важное значение, поскольку может привести к оптимизации кода и заданию более эффективного поведения, особенно если ваше приложение

для Windows 8 работает на мобильных устройствах. Фактически, устройство имеет практически такую же вычислительную мощность, как ноутбук, если брать в расчет производительность вычислений, емкость аккумулятора и объем памяти.

Состояния приложения Магазина Windows

Поскольку Windows 8 является операционной системой, разработанной, в частности, и для мобильных устройств, она не может игнорировать основные правила работы мобильных операционных систем. На мобильных устройствах пользователь запускает приложение, но никогда не завершает его работу. Будучи запущенным, приложение становится своеобразной собственностью операционной системы и его жизненный цикл управляется операционной системой.

Все приложения Магазина Windows могут пребывать в одном из следующих четырех состояний: выполнения, приостановки, завершения или просто не работать. Как показано на рис. 5.8, переходы между этими состояниями определяются пользователем и системой.

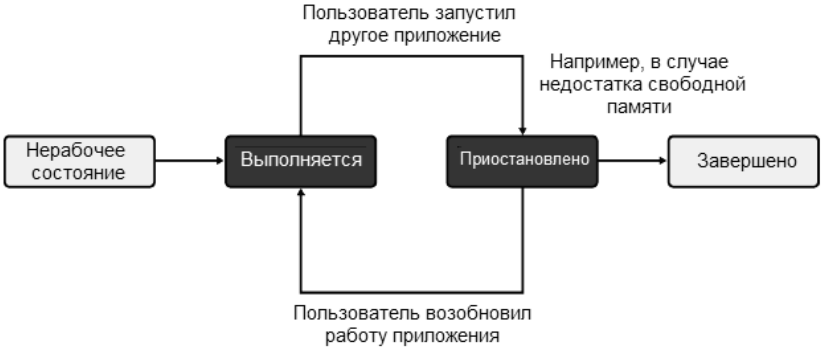

Рис. 5.8. Состояния приложения Магазина Windows

Запуск приложения

У пользователя есть всего несколько способов запуска приложения. Наиболее распространенный способ заключается в том, что пользователь запускает приложение с помощью плитки — аналога значка или ярлыка в Windows 8. Другая возможность запуска приложения появляется, когда пользователь ищет какие-либо данные, выставленные приложением, или обращается к общим с приложением данным. Обе задачи обычно решаются посредством элементов выдвигающейся панели. И наконец, еще одна возможность заключается в запуске приложения из-за открытия пользователем связанного с приложением файла.

Запуск приложения всегда становится причиной возникновения события `activated`, инициируемого библиотекой WinJS, которое можно обработать посредством ранее упомянутого события `onactivated`:

```
app.onactivated = function (args) {
    ...
}
```

В большинстве случаев в ходе события `activated` вам нужно проделать какую-нибудь работу. Например, проверить предыдущее состояние приложения и предпринять соответствующие действия. Показанный далее код является частью любого приложения, основанного на использовании библиотеки WinJS, и представляет собой шаблон, в который можно добавить любой имеющийся у вас код инициализации. Обычно после запуска загружаются исходные данные.

```
app.onactivated = function (args) {
    if (args.detail.kind === activation.ActivationKind.launch) {
       if (args.detail.previousExecutionState !==
           activation.ApplicationExecutionState.terminated) {
          // Сюда помещается код инициализации вашего приложения
       } else {
          ...
       }

       // Включение режима привязки данных (если таковые имеются)
       // на всей странице
       args.setPromise(WinJS.UI.processAll());
    }
};
```

В отношении метода `setPromise` в данном фрагменте кода стоит дать небольшое пояснение. Этот метод информирует приложение о выполнении некой работы в асинхронном режиме. Метод, являющийся хранителем метода `setPromise`, не может быть выполнен, пока не завершится отложенная работа. Но обработка продолжается вплоть до окончания выполнения метода-хранителя. Тем самым возникает проблема надежного принятия решения о том, что делать, когда завершится отложенная работа, называемая также *обязательством* (promise). У WinJS-объекта обязательства имеется метод `then`, используемый для задания любой работы, которая должна быть выполнена в рамках обязательства. Рассмотрим пример:

```
args.setPromise(WinJS.UI.processAll()
    .then(function() { ... })
);
```

Есть еще один похожий метод по имени `done`. Разница между методами `done` и `then` состоит лишь в том, что `then` возвращает еще один объект обязательства, а `done` не возвращает никакого значения:

```
args.setPromise(WinJS.UI.processAll()
    .then(function() { ... })
    .then(function() { ... })
    .done(function() { ... })
);
```

Вместе они выполняют одинаковую работу, но метод `done` может находиться только в конце цепочки действий.

Приостановка выполнения приложения

В Windows 8 на переднем плане может быть активным только одно приложение. Если пользователь переключается на новое приложение, Windows 8 приостанавливает работу выполняемого приложения и переводит его в фоновый режим. Приостановленное приложение остается в памяти, даже если код не выполняется. Когда приложение собираются приостановить, приложение получает событие `checkpoint`. Обработать это событие можно следующим образом:

```
app.oncheckpoint = function (args) {
    // Сохранение данных приложения на случай завершения работы
    WinJS.Application.sessionState["location"] = ...;
};
```

При приостановке выполнения нужно сохранить данные, которые в дальнейшем могут помочь максимально точно восстановить состояние приложения, в котором оно было приостановлено. Для сохранения значений используется словарь `sessionState`. Этот словарь содержит список именованных записей, например, `location`, и получает строковые значения. Названия записей имеют произвольный характер, но обычно они свидетельствуют о роли сохраняемых данных.

Пользователь может переключиться на приостановленное приложение в любое время, при этом Windows просто пробуждает приложение, которое выходит на передний план вместо текущего выполняемого приложения.

Приостановленное приложение кэшируется в памяти на максимально возможное время, но гарантий того, что оно будет там находиться бесконечно, нет. Может случиться, что, к примеру, при дефиците свободной памяти Windows полностью завершит работу приостановленных приложений. В таком случае, если пользователь захочет вернуть приложение на передний план, он сможет только перезапустить его из плитки.

Возобновление выполнения приложения

Разработчику предоставляется довольно простой способ определения, было ли приложение запущено из плитки или из выдвигающейся панели, либо его выполнение было возобновлено после приостановки. Возникает событие `activated`, и для свойства `ActivationKind` устанавливается значение `launch`. Кроме того, если для свойства `previousExecutionState` установлено значение `terminated`, значит, приложение было запущено после приостановки. При возобновлении выполнения приложения после приостановки может понадобиться извлечь и восстановить какие-либо сохраненные данные. Сохраненное состояние обычно извлекается из словаря `sessionState` или из того места, где оно было сохранено при приостановке. Следующий код показывает, куда вставляется код при восстановлении выполнения:

```
app.onactivated = function (args) {
    if (args.detail.kind === activation.ActivationKind.launch) {
       if (args.detail.previousExecutionState !==
           activation.ApplicationExecutionState.terminated) {
```

```
            // Здесь проводится инициализация вашего прилжения
        } else {
            // Здесь восстанавливается состояние приложения
            var data = WinJS.Application.sessionState["location"];
            ...
        }
        args.setPromise(WinJS.UI.processAll());
    }
};
```

Использовать словарь `sessionState` для сохранения данных не обязательно. Данные сеанса можно хранить в каком-нибудь постоянном хранилище или просто посчитать вполне допустимым постоянно перезапускать приложение с чистым состоянием. В основном все зависит от вас, но чаще всего состояние сеанса сохраняется.

Фоновые задачи

В Windows 8, как и на других мобильных платформах, могут выполняться фоновые задачи, не связанные с пользовательским интерфейсом, например задачи передачи данных.

Фоновая задача является облегченным классом, связанным с заданным приложением и запускающимся периодически, пока приложение не запущено. Фоновая задача может быть связана с условием и может не выполняться, пока это условие не будет выполнено.

Фоновая задача также может выводить информацию на экран блокировки. Экран блокировки в Windows 8 содержит фоновое изображение и выводимую на него информацию, например, текущее время, состояние сети и уровень заряда аккумулятора. Кроме того, фоновая задача может выводить на экран блокировки конкретный текст и быстро обновлять его состояние.

Выводы

Этой главой завершается первая часть книги, где вами, по сути, проделана большая работа по закладке фундамента для изучения всего остального материала книги и для вашего путешествия в мир приложений Магазина Windows. Теперь вам уже должны быть знакомы технологии HTML и CSS, кроме того, у вас должны быть достаточные знания языка JavaScript, чтобы разобраться в конкретных WinJS-компонентах, которые встретятся в следующих главах.

6 Пользовательский интерфейс приложений Магазина Windows

Не падай духом — так не годится.

Вильям Шекспир

С этой главы начинается та часть книги, которая посвящена конкретным примерам и упражнениям по программированию для Windows 8. Вы узнаете, как использовать конкретные компоненты и функции операционной системы Microsoft Windows 8, а также изучите ряд приемов создания кода, который не «просто работает», а еще легко читается и хорошо структурирован. В этой главе главное внимание уделено представлению — пользовательскому интерфейсу и связанным с ним аспектами, такими как визуальные компоненты, формы ввода, всплывающие окна и макеты страниц в целом.

ВНИМАНИЕ

Читабельность не относится к тем свойствам кода, которые присущи только творениям специалистов, напротив, стремиться сделать код понятней нужно уже в самом начале карьеры программиста. Когда весь ваш код будет хорошо читаться, польза от этого проявится сразу же, вы это почувствуете, вернувшись к работе над своим кодом после перерыва в несколько дней или недель.

Закладка фундамента приложений Магазина Windows

При открытии Microsoft Visual Studio с намерением создать новое приложение Магазина Windows сначала нужно решить, какого типа проект вы хотите создать. Visual Studio предлагает варианты, называемые шаблонами. Шаблоны, помимо всего прочего, предоставляют вам базовый макет приложения (рис. 6.1).

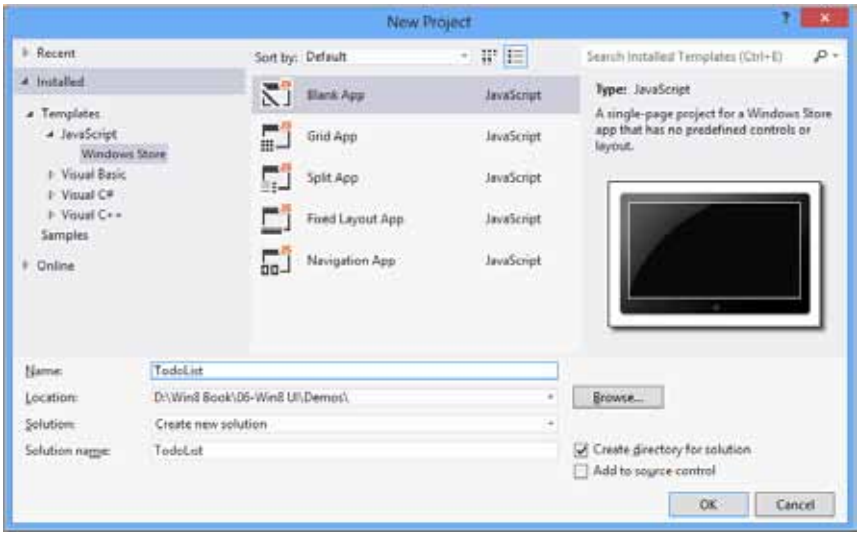

Рис. 6.1. Выбор макета нового приложения

Выбор типа проекта играет важную роль, хотя и не является обязательным. Шаблон в основном служит для «бесплатного» предоставления вам каркаса приложения в соответствии с выбранным вами макетом. Большинство вариантов проектов сводится к тому, что будущее приложение будет состоять либо из одной, либо из нескольких страниц.

Для всех созданных до сих пор учебных приложений использовался шаблон Blank App (Пустое приложение). Этот шаблон состоит всего лишь из пустой HTML-страницы, которую можно заполнить нужным контентом и назначить ей нужное поведение. В этом смысле шаблон Blank App обеспечивает наиболее гибкий подход. И хотя он не предлагает вам сложного макета страницы, вы все же получаете полностью функциональный проект, готовый к компиляции и запуску.

Определение макета приложения

Практически все приложения Магазина Windows требуют от пользователя перемещений между страницами. Иногда приложение предоставляет меню навигации

в явном виде, чтобы пользователи могли быстро находить список доступных страниц и легко возвращаться к нему. Этому сценарию соответствует шаблон Navigation App (Приложение с навигацией), который можно увидеть на рис. 6.1.

В других случаях у приложения нет специально выделенной панели навигации, однако имеется форма навигации, на которой пользователь может щелчком развернуть тот или иной пункт списка, получив более детальное представление. Каркас приложения такого типа вы получите при выборе шаблона Split App (Разделенное приложение) или Grid App (Табличное приложение). Сейчас в качестве шаблона проекта нужно выбрать вариант Blank App, а с примерами на базе шаблона Grid App мы проработаем в следующих главах.

Изучение структуры проекта

Откройте Visual Studio и создайте новый проект типа Blank App, назвав его `TodoList`. Основной целью упражнения является создание формы ввода для сбора информации о некой деятельности с целью ее отслеживания. На рис. 6.2 полностью показано содержимое только что созданного проекта в окне Solution Explorer.

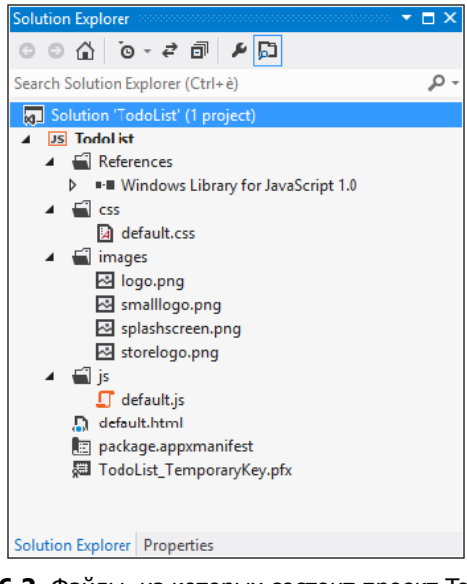

Рис. 6.2. Файлы, из которых состоит проект TodoList

Папка References (Ссылки) содержит список библиотек, требуемых для компиляции приложения. В данном случае единственной необходимой библиотекой является уже известная нам библиотека WinJS.

В папке css содержатся таблицы стилей для стилизации различных HTML-страниц, используемых в вашем приложении. Считается, что на каждую добавляемую к проекту страницу нужно по одному CSS-файлу.

Папка Images (Изображения) является хранилищем всех необходимых вам файлов изображений. Она включает в себя изображения, показываемые пользователю в качестве части интерфейса, а также изображения, необходимые для интеграции приложения в среду Windows 8 и среду Магазина Windows. В табл. 6.1 объясняется, для чего нужны файлы изображений с расширением .png, показанные на рис. 6.2.

Таблица 6.1. Назначение изображений в предлагаемом по умолчанию шаблоне проекта

Файл изображения	Назначение
Logo.png	Изображение, появляющееся на плитке, зарезервированной для приложения на стартовом экране. Это изображение может быть предоставлено либо в формате .png, либо в формате .jpg. Рекомендуемый размер — 150 × 150 пикселов. Windows обычно размещает название приложения поверх изображения, поэтому вам можно не включать имя приложения в изображение
SmallLogo.png	Изображение, предназначенное для представления приложения в результатах поиска. Формат может быть либо .png, либо .jpg, а рекомендуемый размер — 30 × 30 пикселов
SplashScreen.png	Изображение для заставки, показываемой на относительно короткий период времени загрузки приложения после его запуска пользователем. Формат может быть либо .png, либо .jpg, а рекомендуемый размер — 620 × 300 пикселов
StoreLogo.png	Изображение, предназначенное для представления приложения в Магазине Windows. Формат может быть либо .png, либо .jpg, а рекомендуемый размер — 50 × 50 пикселов

Вы можете добавлять и другие изображения, например, WideLogo.png для большого файла логотипа размером до 310 × 150 пикселов. И наконец, файл BadgeLogo.png представляет собой небольшое изображение (обычно 33 × 33 пикселов), выводимое на экране блокировки устройства с системой Windows 8 для идентификации ожидаемых уведомлений от вашего приложения.

В папке js находятся JavaScript-файлы, содержащие всю логику приложения. Обычно в проекте на каждую HTML-страницу приходится один JavaScript-файл, но у вас могут быть также дополнительные JavaScript-файлы, совместно используемые несколькими страницами проекта.

В корневой папке проекта находятся три файла, перечисленные в табл. 6.2.

Таблица 6.2. Назначение файлов в корневой папке шаблона проекта

Файл	Назначение
default.html	HTML-страница, предназначенная для определения главного экрана приложения
Package.appxmanifest	Вся информация, необходимая для пакетирования вашего приложения Магазина Windows с целью его распространения
*Xxx*_TemporaryKey.pfx	Временный сертификат, автоматически выдаваемый для тестирования приложения на машине для разработки. Символы *Xxx* заменяют реальное имя приложения. С помощью сертификата подписывается любой исполняемый файл, появившийся в результате выполнения проекта. Когда разработка приложения завершится, вам нужно будет заменить этот тестовый сертификат реальным, получаемым из учетной записи Магазина Windows. Получение реального сертификата является необходимым для публикации вашего приложения в общедоступном магазине

Это минимальный набор файлов, необходимых в приложении Магазина Windows. По мере создания приложения в проекте обычно приходится создавать собственные нестандартные папки для дополнительных файлов всех типов, поскольку нужно больше HTML-страниц, таблиц стилей, JavaScript-файлов и изображений.

Следующий шаг упражнения заключается во внесении некоторых изменений в базовый пользовательский интерфейс с целью получения формы, определяющей элементы, которые войдут в список текущих дел (to-do list).

Изучение стандартного стиля и ссылок на сценарии

Если дважды щелкнуть на файле default.html, открыв его для редактирования, будет видна следующая разметка:

```
<!-- WinJS references -->
  <link href="//Microsoft.WinJS.1.0/css/ui-dark.css" rel="stylesheet" />
  <script src="//Microsoft.WinJS.1.0/js/base.js"></script>
<script src="//Microsoft.WinJS.1.0/js/ui.js"></script>
```

В соответствии с комментарием в первой строке — это не простые ссылки на внешние источники, которые можно найти практически в любой HTML-странице, а специальные ссылки на файлы таблиц стилей и сценариев, изначально входящие в библиотеку WinJS. Ни один из этих файлов не будет доступен в качестве исходного кода проекта, поскольку эти файлы извлекаются из библиотеки WinJS во время выполнения.

Следует заметить, что вы можете дать своему приложению Магазина Windows общую облегченную тему, просто заменив элемент `link` в показанном коде следующим элементом:

```
<link href="//Microsoft.WinJS.1.0/css/ui-light.css" rel="stylesheet" />
```

Ссылка на любой дополнительный CSS-файл или файл сценария, на который вы решите сослаться, должна располагаться в следующем разделе:

```
<!-- TodoList references -->
 <link href="/css/default.css" rel="stylesheet" />
<script src="/js/default.js"></script>
```

В данном случае `/css` и `/js` являются ссылками на физические папки в вашем текущем проекте.

Добавление фиксированных блоков пользовательского интерфейса

Простое приложение, получаемое из шаблона Blank App, имеет темный фон и просто выводит на экран некий текст, который следует заменить нужным вам текстом. Вы можете так или иначе модифицировать свое приложение, например, добавив заголовок и колонтитул. В предыдущих примерах использовались следующие заголовок и колонтитул:

```
<header>
   Start Here! Build <b>Windows 8</b> Applications with <b>HTML5</b> and
 <b>JavaScript</b>
   <hr />
 </header>

 <footer>
   <hr />
   Dino Esposito | Francesco Esposito
</footer>
```

В качестве упражнения давайте сделаем эти компоненты многократно используемыми, чтобы вы могли сохранить их на странице, чтобы ссылаться на нее везде, где это потребуется, не переживая насчет внутренних деталей.

Первый шаг заключается в создании новой нестандартной папки проекта. Нужно щелкнуть правой кнопкой мыши на узле проекта (мы назвали его TodoList) и в появившемся меню выбрать пункт Add ▸ New Folder (Добавить ▸ Новую папку). Назовите новую папку Pages (Страницы).

ПРИМЕЧАНИЕ

Если при создании новой папки вы случайно пропустите этап редактирования имени папки, в проекте появится новая папка по имени New Folder. Но не стоит волноваться: достаточно щелкнуть на этом имени с задержкой (это разновидность длинного щелчка, при которой кнопка мыши удерживается нажатой около секунды). Тогда данный пункт проекта снова перейдет в режим редактирования, и вы сможете просто набрать имя Pages и щелкнуть за пределами текстового поля, чтобы сохранить свои изменения.

Теперь, чтобы добавить многократно используемый блок, нужно щелкнуть правой кнопкой мыши на узле Pages (Страницы) в Visual Studio Solution Explorer и выбрать

в контекстном меню пункт Add ▸ New Item (Добавить ▸ Новый элемент). Затем в окне, показанном на рис. 6.3, нужно выбрать вариант HTML Page (HTML-страница) и назвать страницу header.html. Далее нужно повторить эти шаги, создав вторую HTML-страницу с именем footer.html.

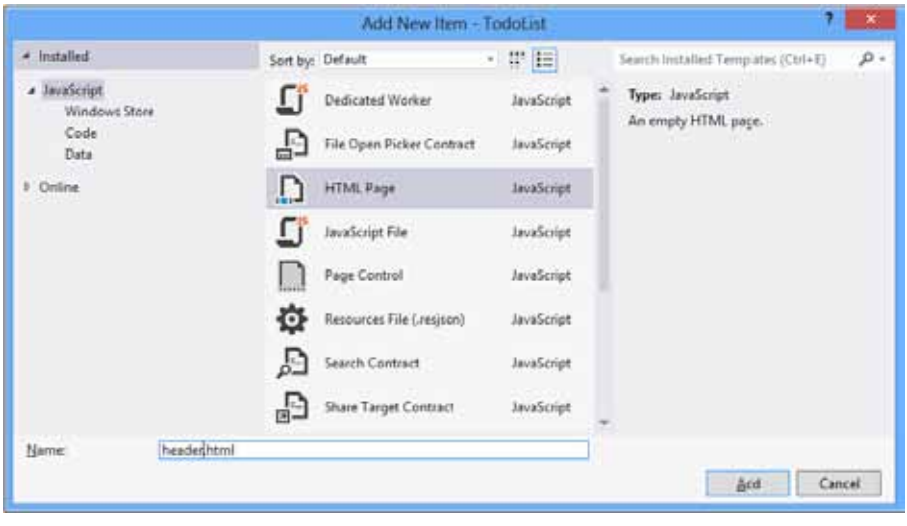

Рис. 6.3. Добавление нового компонента (в данном случае — HTML-страницы)

Содержимое обоих только что добавленных HTML-файлов одинаковое:

```
<!DOCTYPE html>
 <html>
    <head>
       <title></title>
    </head>
    <body>
    </body>
</html>
```

Отредактируйте содержимое файла header.html следующим образом:

```
<!DOCTYPE html>
 <html>
    <head>
       <title></title>
    </head>
    <body>
       <header>
          Start Here! Build <b>Windows 8</b> Applications with <b>HTML5</b> and
          <b>JavaScript</b>
          <hr />
       </header>
    </body>
</html>
```

Затем отредактируйте содержимое файла footer.html:

```
<!DOCTYPE html>
 <html>
     <head>
         <title></title>
     </head>
     <body>
     <footer>
         <hr />
         Dino Esposito | Francesco Esposito
     </footer>
     </body>
</html>
```

Сохраните внесенные изменения.

Следующий шаг состоит в создании ссылок из главной страницы default.html на ваши новые элементы заголовка и колонтитула. Откройте файл default.html для редактирования и внесите показанные ниже изменения в тег body:

```
<body>
    <div data-win-control="WinJS.UI.HtmlControl"
         data-win-options="{uri:'/pages/header.html'}"></div>
    <h1>TO-DO List</h1>
    <div data-win-control="WinJS.UI.HtmlControl"
         data-win-options="{uri:'/pages/footer.html'}"></div>
</body>
```

Атрибут `data-win-control` превращает элемент `div` в HTML-элемент управления, предназначенный просто для вывода содержимого файла, на который дается ссылка, то есть файла header.html. То же самое происходит и с файлом footer.html.

Ранее в главе 1 мы включили в файл default.css ряд цветовых и стилевых параметров; сделаем то же самое и здесь, изменив файл default.css следующим образом:

```
body {
     background-color: #1649AD;
     padding: 10px;
 }
 header {
     font-size: x-large;
     color: #ffffff;
     padding-bottom: 50px;
 }
 footer {
     padding-top: 50px;
     font-size: large;
     color: #eeee00;
     font-weight: bold;
}
```

Файл default.css содержит стилевую информацию для страницы default.html. Если нужно многократно использовать эти стилевые параметры на другой странице,

например на странице mypage.html, параметры из файла default.css можно скопировать в новый файл с именем mypage.css. Этот способ вполне приемлем, но должен существовать и более удобный способ получения тех же результатов (и такой способ есть).

Щелкните правой кнопкой мыши на папке css проекта и выберите пункт Add ▸ New Item (Добавить ▸ Новый элемент). В следующем окне, показанном на рис. 6.4, выберите пункт Style Sheet (Таблица стилей) и назовите его по имени проекта, в данном случае — todolist.css. Скопируйте в ваш новый файл todolist.css содержимое CSS-классов `body`, `header` и `footer`.

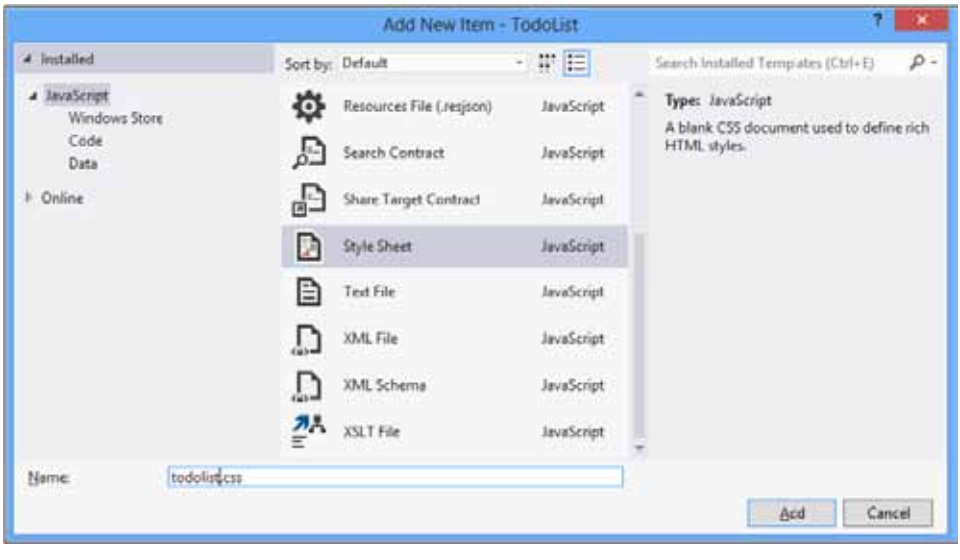

Рис. 6.4. Добавление собственного нестандартного CSS-файла

Теперь у вас есть глобальный CSS-файл, где планируется хранить все глобальные стилевые параметры, кроме того, у вас есть CSS-файл со стилевыми параметрами только для одной конкретной страницы. Чтобы глобальный CSS-файл был видим любой странице, во все страницы, представляющие экран приложения, нужно добавить еще одну строку кода. Пока это нужно сделать только для страницы default.html:

```
<!-- TodoList references -->
  <link href="/css/todolist.css" rel="stylesheet" />
<link href="/css/default.css" rel="stylesheet" />
```

Теперь можно запустить приложение, оно должно выглядеть так, как показано на рис. 6.5. И хотя это по-прежнему довольно убогое приложение, вы, по крайней мере, освоили два очень полезных приема, позволяющие многократно использовать разметку и стили.

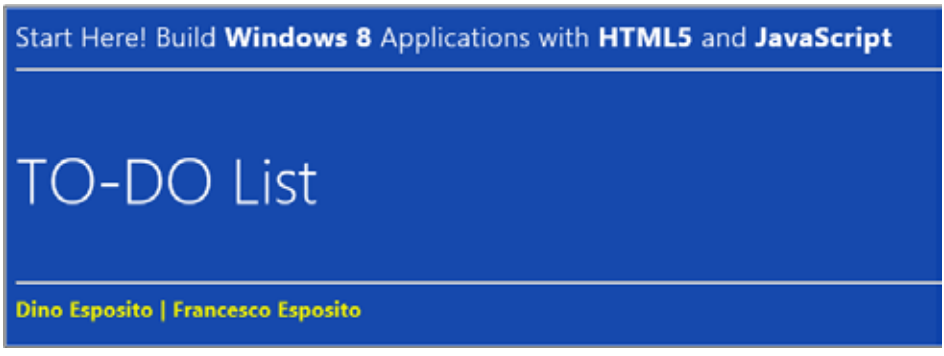

Рис. 6.5. Текущий вид приложения TodoList

Атрибуты приложения

Перед добавлением к учебному приложению средств взаимодействия с ним нужно потратить некоторое время на его конфигурирование — эта операция не является особо сложной, но она требуется всем приложениям Магазина Windows, поэтому кратко поговорим о манифесте, логотипах и заставке.

Файл манифеста

Visual Studio создает файл манифеста автоматически при создании нового проекта приложения Магазина Windows. Этот файл содержит имя приложения и описание, сведения о логотипах и другую базовую информацию, например сведения о стартовой странице приложения. Файл манифеста изначально имеет ряд предлагаемых по умолчанию значений, которые в некоторый момент, возможно, потребуется изменить. Visual Studio предоставляет удобный редактор, облегчающий внесение требуемых изменений. На рис. 6.6 показано окно редактора, для открытия которого достаточно дважды щелкнуть на файле package.appxmanifest в папке проекта.

Требуемые атрибуты сгруппированы по функциональным областям. Сейчас мы сосредоточимся на вкладке Application UI (Пользовательский интерфейс приложения). В поле Display name (Выводимое имя) указывается официальное имя приложения в том виде, в котором оно будет присутствовать в Магазине Windows и в меню Windows 8, например, на стартовом экране. В поле Start page (Стартовая страница) по умолчанию указан вариант default.html; чтобы изменить этот параметр, нужно просто создать новую страницу или переименовать существующую, а затем соответствующим образом изменить манифест. В полях Default language (Язык по умолчанию) и Description (Описание) можно указать язык, предлагаемый по умолчанию, и описание. Далее в разделе Supported rotations (Поддерживаемые варианты поворота) можно указать, какие повороты поддерживаются (если приложение выполняется на планшетном компьютере), а в разделе Tile (Плитка) — выбрать изображения для логотипов.

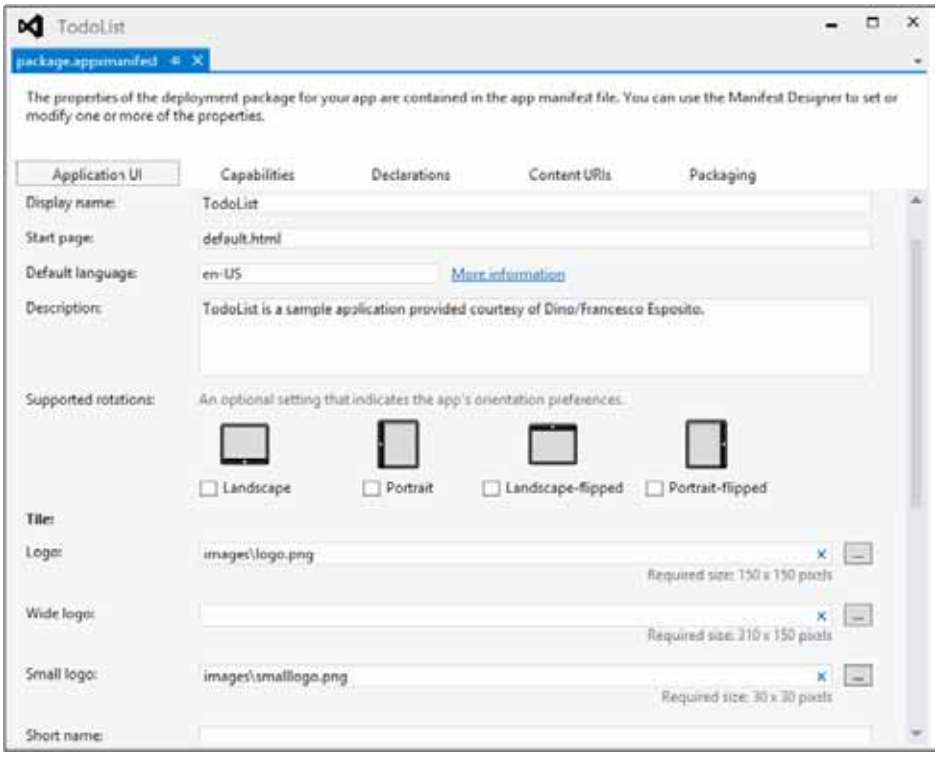

Рис. 6.6. Редактор файла манифеста

Файл манифеста содержит также техническую информацию о потребностях приложения, например, нужен ли ему доступ к локальному хранилищу или к веб-камере. Эти дополнительные параметры задаются на других вкладках, показанных на рис. 6.6, Capabilities (Возможности), Declarations (Объявления) и т. д. Мы поработаем с ними в следующих главах при разработке конкретных механизмов приложения, требующих специальной настройки манифеста.

Добавление изображений логотипов

Любое приложение Магазина Windows нуждается в наборе изображений, используемых в различных сценариях для быстрой и простой идентификации приложения. В самом минимальном варианте приложению для Windows 8 нужны четыре изображения (см. табл. 6.1). При подготовке этих изображений нужно следовать ряду рекомендаций.

В первую очередь, для логотипа Windows 8 используется прозрачный фон. Это означает, что сам логотип состоит из рисунка, помещенного в прямоугольную область. Фоновая область делается прозрачной с помощью такого специализированного графического инструмента, как Paint.NET, который является бесплатным

средством редактирования изображений, получаемым на веб-сайте http://www.getpaint.net. На рис. 6.7 показан логотип учебного приложения TodoList, помещенный на прозрачный фон.

Текстура в виде шахматки, показанная на рисунке, является принятым в Paint.NET способом сообщить вам, что у вашего изображения прозрачный фон. И хотя задавать файлам логотипа прозрачный фон не обязательно, благодаря ему ваши изображения логотипа соответствуют общему виду пользовательского интерфейса Windows 8 (и, в любом случае, становятся более симпатичными).

Есть еще один небольшой трюк, который можно применять к изображениям для логотипа; в результате логотип будет больше походить на логотипы некоторых «родных» для Windows 8 приложений. Этот трюк заключается в том, чтобы сделать изображение плоским и дать ему единственный цвет — белый. В конечном итоге логотип получается белым на прозрачном фоне. И хотя на этапе редактирования он выглядит, очевидно, не слишком впечатляюще, ситуация серьезно улучшается при установке приложения в среду Windows 8. На рис. 6.8 показано установленное приложение TodoList с белым логотипом на прозрачном фоне.

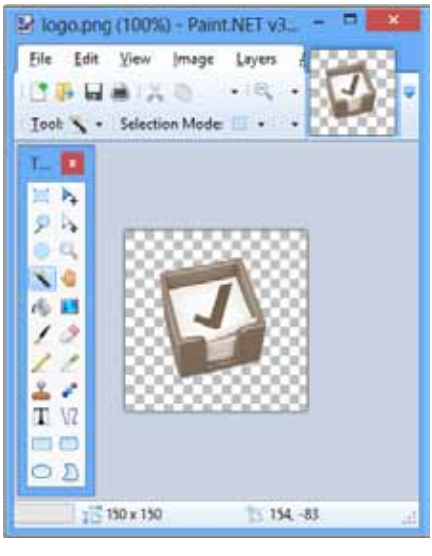

Рис. 6.7. Добавление прозрачного фона к изображению логотипа

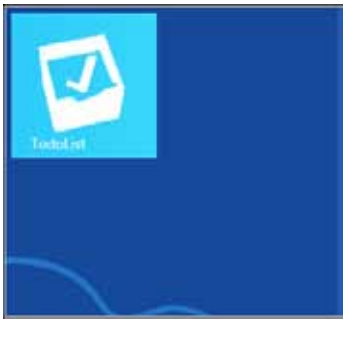

Рис. 6.8. Логотип приложения TodoList, выполненный в белом цвете на прозрачном фоне. Имя приложения система добавляет автоматически

Осталось еще рассмотреть фоновый цвет плитки приложения в стартовом окне Windows 8. Как уже упоминалось, для изображения предлагается прозрачный фон, но цвет плитки можно задать. Это делается путем вода цвета в формат HTML в файле манифеста. Если немного прокрутить вниз окно, показанное на рис. 6.6, вы

увидите в нем поле Background (Фон). Как мы выяснили в главе 2, цветовые параметры в HTML выражаются в формате *#rrggbb*, где *rr*, *gg* и *bb* — шестнадцатеричные значения для компонентов красного, зеленого и синего цветов. Для получения красивого светло-синего цвета введите в это поле значение `#2eccfa`.

Добавление изображения для заставки

У каждого приложения Магазина Windows должна быть заставка, то есть изображение, которое появляется сразу же после запуска приложения пользователем и остается видимым до тех пор, пока не будет готово к работе. В любом шаблоне проекта Visual Studio предлагается пустой экран заставки, который разработчикам остается только отредактировать в таком средстве работы с графикой, как, например, Paint.NET. На рис. 6.9. показан переработанный экран заставки для приложения TodoList.

Рис. 6.9. Экран заставки для учебного приложения TodoList

Существует ряд принципов, которых нужно придерживаться при создании заставок. Для начала нужно запомнить, что единственным назначением заставки является предоставление пользователю немедленной ответной реакции от приложения. В руководстве для Windows 8 рекомендуется делать для заставки очень простые изображения; такое изображение обычно состоит из логотипа приложения, помещенного в центр поверхности размером 620 × 300 пикселов.

В идеале логотип должен иметь прозрачный фон и использовать цвета, хорошо различимые на выбранном для экрана заставки фоне. Задать фоновый цвет для экрана заставки можно в редакторе манифеста. На экране заставки лучше также не показывать рекламной информации и информации о версии приложения.

Начало серьезной работы над приложением TodoList

Следующий шаг в процессе создания приложения TodoList состоит в добавлении интерактивной формы, дающей пользователю возможность вводить информацию, касающуюся некоего задания, и отслеживать ход его выполнения. В процессе работы вы познакомитесь с некоторыми элементами управления вводом, доступными в Windows 8, а также с некоторыми HTML5-элементами, уже встречавшимися в главе 2.

Создание интерактивной формы

Форма ввода делится на четыре раздела, каждый из которых представлен элементом `div`. Откройте в Visual Studio файл default.html и внесите в элемент `body` следующую разметку:

```
<body>
    <div data-win-control="WinJS.UI.HtmlControl"
         data-win-options="{uri:'/pages/header.html'}"></div>
    <h1>TO-DO List</h1>

    <div class="form-container">
       <div class="form-section"> ... </div>
       <div class="form-section"> ... </div>
       <div class="form-section"> ... </div>
       <div class="form-section"> ... </div>
    </div>

    <div data-win-control="WinJS.UI.HtmlControl"
         data-win-options="{uri:'/pages/footer.html'}"></div>
</body>
```

Блок `div` с CSS-классом по имени `form-container` является просто блоком разметки, который мы будем заполнять в остальной части главы. Добавьте в файл default.css следующий код, предназначенный для придания стиля элементам контейнера `div`:

```
.form-container {
    border-radius: 10px;
    background-color: #99e;
    color: #eee;
    padding: 20px;
    margin: 5px;
    width: 600px;
 }
 .form-section {
    margin-top: 10px;
 }
 .block-element {
    display: inline-block;
    vertical-align: top;
}
```

В самом верхнем элементе `div` помещаются элементы управления, предназначенные для ввода описания задания и срока его выполнения. Во втором элементе `div` выводится приоритет задания, а третий элемент дает пользователю возможность задать статус задания. И наконец, в четвертом элементе `div` есть кнопка для сохранения задания.

Определение объекта задания

Основная часть JavaScript-кода, вставляемого в HTML-страницы, касается обновления элементов. Привязка данных является эффективным приемом минимизации соответствующего кода, при этом основная часть работы возлагается на нижележащую инфраструктуру. Чтобы привязка данных работала незаметно, нужно создать объект, как можно точнее соответствующий данным, планируемым к чтению и записи посредством формы. В данном случае вам нужен объект `Task`. Для его создания добавьте в папку js проекта новый JavaScript-файл. Его можно назвать по имени приложения — todolist.js.

Перед тем как приступить к редактированию файла todolist.js, нужно внести небольшие изменения в файл default.js, чтобы получить возможность контролировать фазу загрузки приложения. Эти изменения аналогичны тем, которые вносились в код в рассмотренном в предыдущей главе примере привязки данных. И вообще, это изменение требуется вносить в любое приложение Магазина Windows. Нужно найти функцию `app.onactivated` и заменить строку с вызовом метода `setPromise` следующим кодом:

```
args.setPromise(WinJS.UI.processAll()
                  .then(TodoList.init())
);
```

В результате после завершения фазы активации приложения этот код уступит управление функции по имени `TodoList.init`.

Теперь обратите внимание на только что созданный файл todolist.js и добавьте к нему следующий код:

```
var TodoList = TodoList || {};

 TodoList.init = function () {
     TodoList.performInitialBinding()
 }

 TodoList.Priority = {
     VeryLow: 1,
     Low: 2,
     Normal: 3,
     High: 4,
     VeryHigh: 5
 };

 TodoList.firstOfNextMonth = function () {
```

```
    var d = new Date();
    d.setDate(d.getDate() + 31);
    var year = d.getFullYear();
    var month = d.getMonth();
    var day = 1;
    var newDate = new Date(year, month, day);
    return newDate;
 }
 // Определение объекта Task
    var Task = WinJS.Class.define(function () {
    var that = {};
    that.description = "This is my new task";
    that.dueDate = TodoList.firstOfNextMonth();
    that.priority = TodoList.Priority.Normal;
    that.status = "Not Started";
    that.percCompleted = 0;
    that.minPriority = TodoList.Priority.VeryLow;
    that.maxPriority = TodoList.Priority.VeryHigh;
    return that;
});
```

Теперь у вас есть глобальный объект по имени `TodoList`, в котором содержатся все JavaScript-функции и JavaScript-объекты, используемые приложением. Кроме того, у вас есть объект `Task` с несколькими свойствами задания, таким как описание (`description`), срок выполнения (`dueDate`), приоритет (`priority`), состояние (`status`) и процент завершения (`percCompleted`). Далее нужно будет отредактировать код пользовательского интерфейса, чтобы он обеспечивал сбор данных для заполнения нового объекта `Task`. Одновременно вам может потребоваться использовать существующий экземпляр объекта `Task` для инициализации полей ввода.

Сбор текста и данных

В самый верхний элемент `div` нужно поместить многострочный текстовый редактор и средства ввода даты. Многострочный текстовый редактор требует HTML-элемента `textarea`. Для ввода даты есть два средства. Можно воспользоваться HTML-элементом ввода или Windows-компонентом выбора даты. Windows-компонент выбора даты существенно упрощает взаимодействие с пользователем. Разметка, которую нужно вставить в первый элемент `div` с классом `form-section`, имеет следующий вид:

```
<div class="form-section">
    <div class="block-element">
        <h3>DESCRIPTION</h3>
        <textarea id="taskSubject" required
                  data-win-bind="innerText: description"></textarea>
    </div>
    <div class="block-element">
        <h3>DUE DATE</h3>
        <div class="block-element" id="taskDueDate"
             data-win-control="WinJS.UI.DatePicker" />
    </div>
</div>
```

Давайте на время отвлечемся от атрибута `data-win-bind` и в целом от темы привязки данных, обратив внимание на атрибут `data-win-control`, используемый в теге `div` с атрибутом `id`, имеющим значение `taskDueDate`. Атрибут `data-win-control` превращает обычный элемент `div` в компонент выбора даты в Windows 8. Поэтому, хотя в разметке вы видите обычный элемент `div`, на самом деле в процессе выполнения программы вы получаете гораздо более сложное дерево элементов.

Установка приоритета задания

Приоритет задания выражается числом в диапазоне от единицы (очень низкая) до пяти (очень высокая). Вам нужно просто заставить пользователя ввести число в текстовое поле. Для этого достаточно простого поля ввода. В Windows 8 можно использовать один из новых типов элементов ввода, предлагаемых спецификацией HTML5. Далее среда размещения автоматически выведет элемент ввода в виде ползунка. Для этого понадобится только следующая разметка, которая займет вторую область вашей формы ввода:

```
<div class="form-section">
     <div class="block-element">
         <h3>PRIORITY (1=VERY LOW - 5=VERY HIGH)</h3>
         <input id="taskPriority" type="range"
                 data-win-bind="value: priority; min: minPriority; max:
   maxPriority"
     </div>
</div>
```

Для атрибута `type`, принадлежащего элементу `input`, установлено строковое значение `range`, которое позволяет предложить пользователям красивый ползунок. Исходное значение ползунка, а также его минимальное и максимальное значения устанавливаются путем привязки данных. Детали такой привязки данных мы рассмотрим позднее.

Установка статуса задания

Третья область формы содержит информацию о статусе задания. Статус задается с помощью строки, выбираемой в раскрывающемся списке. Кроме того, вы можете показать процент уже проделанной работы в числовом поле ввода. Для этого нужна следующая разметка:

```
<div class="form-section">
     <div class="block-element">
         <h3>STATUS</h3>
         <select id="taskStatus">
             <option>Not Started</option>
             <option>In progress</option>
             <option>Completed</option>
         </select>
     </div>
     <div class="block-element">
         <h3>% COMPLETED</h3>
```

```
        <input id="taskPercCompleted" type="number" min="0" max="100"
               data-win-bind="value: percCompleted" />
    </div>
</div>
```

Windows 8 не предлагает для раскрывающегося списка какие-либо специальные средства. Обычного HTML-элемента `select` будет вполне достаточно. Присвоение значения `number` атрибуту `type` элемента `input` заставляет поле ввода принимать только числа. Тем не менее имейте в виду, что Windows 8 не мешает вводить также нечисловые символы, которые при считывании фактического значения для дальнейшей обработки просто отбрасываются.

Добавление кнопки и подсказок

Завершающая область формы содержит кнопку, с помощью которой пользователь собирает все введенные данные и запускает операцию, физически сохраняющую где-нибудь эти данные. В этом примере мы не будем ничего делать для сохранения данных, но вы узнаете, как выполняется сбор данных и их объединение для предоставления результатов пользователю. Следующая разметка для четвертой области содержит две интересные вещи: кнопку и подсказку (подсказка представляет собой небольшое всплывающее окно с полезной для пользователя информацией, появляющееся, когда указатель мыши оказывается на элементе управления):

```
<div class="form-section">
    <div>
        <button id="buttonAddTask"
            data-win-control="WinJS.UI.Tooltip"
            data-win-options="{innerHTML: '<b>Purpose</b><hr>Add
            the <i>newly created</i> task to the list.'}">
            Add Task
        </button>
    </div>
</div>
```

ВНИМАНИЕ

В показанном примере содержимое свойства innerHTML разбито на две строки с целью сделать разметку понятнее. В редакторе Visual Studio этот код нужно вводить в виде непрерывной строки, в противном случае он не будет компилироваться.

ПРИМЕЧАНИЕ

В этой книге приложение для Windows 8 создается с помощью HTML и других технологий, связанных с веб-программированием, таких как CSS и JavaScript. В связи с этим у некоторых разработчиков, имеющих определенный опыт в веб-программировании, может вызвать удивление тот факт, что веб-форма создается без HTML-элементов form и submit. В Windows 8 элемент form просто не нужен, поскольку здесь отсутствует серверный компонент, получающий контент формы. В любой форме, предназначенной для сбора вводимых данных, вам нужна лишь одна или несколько кнопок для инициирования действий. Такие кнопки выводятся на экран с помощью обычного элемента button.

В Windows 8 подсказку можно назначить любому HTML-элементу простым добавлением атрибута `data-win-control` со значением `WinJS.UI.Tooltip`. Для инициализации нового элемента управления используется также атрибут `data-win-options`. В данном случае свойству `innerHTML` передается размеченный текст, визуализируемый при наведении указателя мыши на кнопку.

Кнопки не приносят никакой пользы, пока с ними не связаны события щелчка. По сложившейся практике связывать обработчики событий с HTML-элементами лучше всего при инициализации страницы. Для этого нужно вернуться к файлу todolist.js и отредактировать функцию `TodoLit.init` следующим образом:

```
TodoList.init = function () {
     document.getElementById("buttonAddTask")
                 .addEventListener("click", TodoList.addTaskClick);
     TodoList.performInitialBinding()
}
```

Теперь, когда пользователь щелкнет на кнопке Add Task (Добавить задание), начнет выполняться код, определенный в функции `TodoList.addTaskClick`. Этот код также нужно добавить к файлу todolist.js:

```
TodoList.addTaskClick = function () {
     TodoList.alert("Add Task button clicked");
 }
 TodoList.alert = function (message) {
     var alertDialog = new Windows.UI.Popups.MessageDialog(message);
     alertDialog.showAsync();
}
```

Сейчас при щелчке на кнопке будет лишь появляться окно с сообщением для пользователя, но чуть позже мы перепишем этот код для вывода сводных данных о сохраняемом задании.

А что, если вам захочется добавить подсказку к элементу, который уже был преобразован в Windows-компонент с помощью атрибута `data-win-control`? Например, как добавить подсказку к компоненту выбора даты? Рассмотрим следующий пример кода:

```
<div data-win-control="WinJS.UI.Tooltip"
      data-win-options="{innerHTML: 'Specify the due for the task'}">
     <h3>DUE DATE</h3>
     <div class="block-element" id="taskDueDate"
         data-win-control="WinJS.UI.DatePicker" />
</div>
```

Как показано в этой разметке, вам нужно лишь воспользоваться оболочкой в виде элемента `div`, который настроен так, чтобы во время выполнения превращаться в подсказку.

Помещение данных в форму

Пока вы определили только разметку формы, а теперь пора задуматься о том, как привязать ее к данным.

Инициализация формы ввода

В коде инициализации страницы у вас уже есть вызов функции по имени `performInitialBinding`. Нужно еще добавить код к телу этой функции. От нее ожидается весьма простое поведение: функции нужно передать объект `Task` и связать его содержимое с элементами пользовательского интерфейса. Если назначение формы в приложении состоит в добавлении нового задания, можно передать только что созданный (иными словами — пустой) экземпляр объекта `Task`, заполненный исходными значениями.

Однако если форма служит для редактирования существующих (ранее созданных) заданий, код должен сначала извлечь задание для редактирования, загрузить данные в свежий экземпляр объекта `Task`, а затем показать его с помощью пользовательского интерфейса.

```
TodoList.performInitialBinding = function () {
     // Это также может быть объект Task, извлеченный из хранилища
     var task = new Task();

     // Активирование привязки HTML-элемента (элементов) выбора
     var bindableElement = document.getElementById("form-container");
     WinJS.Binding.processAll(bindableElement, task);

     // Установка даты в в компоненте выбора даты
     var datePicker = document.getElementById("taskDueDate").winControl;
     datePicker.current = task.dueDate;

     // Выбор статуса задания в раскрывающемся списке
     var dropDownList = document.getElementById("taskStatus");
     dropDownList.selectedIndex = TodoList.getIndexFromStatus(task.status);
}
```

Привязка данных работает в каскадном режиме, то есть вы прикрепляете объект источника данных к элементу контейнера, а затем используете библиотеку WinJS, чтобы она разрешила за вас все *зависимости* (dependencies). Вы устанавливаете зависимость между элементом пользовательского интерфейса и свойством в источнике данных посредством атрибута `data-win-bind`. Это можно сделать через атрибуты разметки, в таком случае привязка данных называется *декларативной* (declarative data binding). Ее мы сначала и рассмотрим. (Вскоре будет показано, что привязку можно также выполнить не декларативным, а программным способом.)

В предыдущем коде показано, как источник данных (только что созданный объект `Task`) привязывается к элементу `div`, содержащему всю форму. Именно к этому элементу `div` и осуществляется привязка. Создание ссылки требует вызова функции `processAll`.

К сожалению, в WinJS декларативную привязку данных полностью поддерживают не все компоненты. Привязка данных хорошо работает при привязке данных к обычным HTML-элементам. Если у вас более амбициозные цели, например привязка срока выполнения в объекте `Task` к дате, выводимой в компоненте выбора даты, то декларативная привязка данных поддерживается не полностью. Точно так же декларативная привязка данных не поддерживается в отношении обычных раскрывающихся HTML-списков, причем Windows 8 не предлагает таким спискам никакой альтернативы.

Что это означает для разработчиков?

Ответ предельно прост: разработчикам придется писать дополнительный код, чтобы привязать данные к элементам пользовательского интерфейса программным способом. Показанные далее строки из `TodoList.performInitialBinding` показывают, как программным способом заставить компонент выбора даты показать конкретную дату и как программным способом выбрать пункт в раскрывающемся списке, использовав свойство `selectedIndex`:

```
// Установка даты для компонента выбора даты
  var datePicker = document.getElementById("taskDueDate").winControl;
  datePicker.current = task.dueDate;

  // Установка статуса для раскрывающегося списка
  var dropDownList = document.getElementById("taskStatus");
dropDownList.selectedIndex = 1; // Выбор второго элемента
```

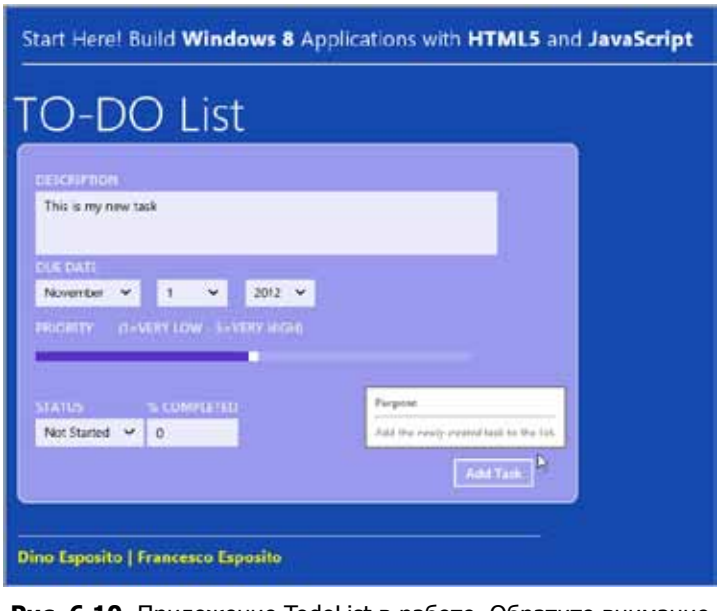

Рис. 6.10. Приложение TodoList в работе. Обратите внимание на подсказку для кнопки Add Task

Теперь, когда все параметры заданы, все готово к созданию и отладке приложения. Результат должен быть похож на то, что показано на рис. 6.10. Если этого не случилось, внимательно изучите сообщения об ошибках, которые передаются либо компилятором, либо средой выполнения программы.

СОЗДАНИЕ И ОТЛАДКА ПРИЛОЖЕНИЯ

Исходный код приложения пишется в контексте решения в Visual Studio, являющегося репозитарием (хранилищем) всех ресурсов, необходимых для работы приложения (исходных файлов, изображений, таблиц стилей, манифеста и т. д.).

Решение в Visual Studio — не то же самое, что приложение, но из решения можно создать исполняемое приложение путем успешной «компоновки» (компиляции) решения. Чтобы скомпоновать текущее решение в Visual Studio, нужно нажать клавишу F7 или выбрать пункт Build Solution (Скомпоновать решение) в меню Build (Компоновка), как показано на рис. 6.11. В процессе компоновки среда Visual Studio вызывает компилятор и обрабатывает исходный код. Если все в порядке и синтаксические ошибки не обнаружены, вы получаете исполняемое приложение, которое можно запустить.

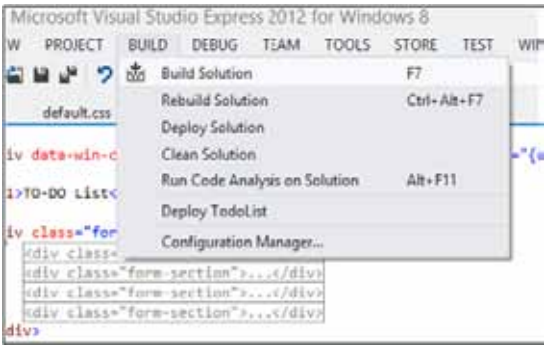

Рис. 6.11. Меню компоновки

Если в процессе компоновки Visual Studio обнаружит ошибки, они будут перечислены в окне, в нижней части экрана среды Visual Studio. Однако если создается JavaScript-приложение (как в данной книге), вы вряд ли получите сообщения об ошибках в ходе компоновки. Скорее всего, вы получите сообщения об ошибках времени выполнения.

Сообщения об ошибках времени выполнения выводятся в ходе отладки, которая заключается в пробном запуске приложения с целью проверить, что оно работает так, как от него ожидалось. Отладка приложения инициируется нажатием клавиши F5 или выбором пункта Start Debugging (Запуск отладки) в меню Debug (Отладка).

На рис. 6.12 показаны последствия синтаксической ошибки: пропуска запятой непосредственно перед выделенной строкой. Учтите, что получаемое сообщение может ввести в заблуждение относительно реальной причины проблемы. На рисунке, как может показаться, Visual Studio указывает на отсутствие в коде закрывающей скобки. Конечно, и это возможно, но реальная проблема, скорее всего, не в этом. То есть, несомненно, обращая внимание на сам факт наличия ошибки, нужно относиться скептически к конкретным сообщениям об ошибках, поскольку они являются «автоматически выстроенными догадками» тех или иных программных компонентов и не всегда отражают истинную причину проблемы.

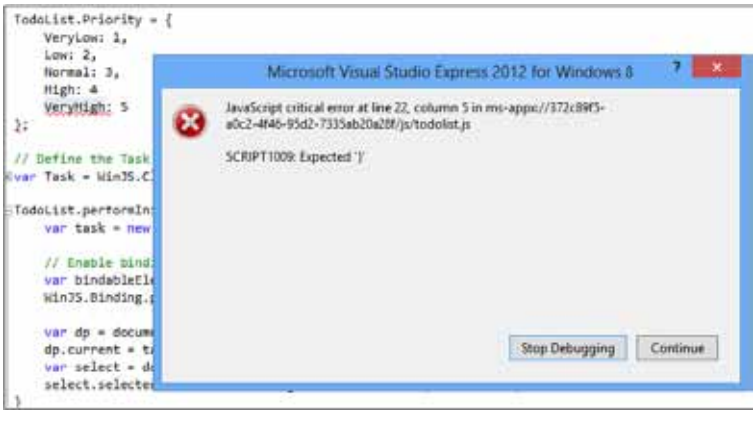

Рис. 6.12. Обнаружена синтаксическая ошибка

Также при отладке часто возникает исключение. Исключение отличается от синтаксической ошибки тем, что в этом случае код имеет правильное синтаксическое построение, но неверен логически, из-за чего происходит сбой во время выполнения программы при обработке фактических данных.

Исключения часто происходят из-за данных, которые либо неправильно введены пользователем, либо неверно сгенерированы программным кодом. На рис. 6.13 показано исключение, возникшее из-за попытки использования неопределенной переменной.

Всплывающие окна (подобные показанному на рис. 6.13) появляются, когда система обнаруживает исключение. При этом выполнение останавливается щелчком на кнопке Break (Прервать), а по щелчку на кнопке Continue (Продолжить) можно попытаться заставить приложение продолжить работу. Чаще всего требуется прервать работу приложения, выяснить причину его нештатной работы и внести исправления в код. Окно Output (Вывод) отладчика, которое обычно располагается в нижней части окна Visual Studio, в любое время позволяет получить доступ к отчету об обнаруженных ошибках приложения (рис. 6.14).

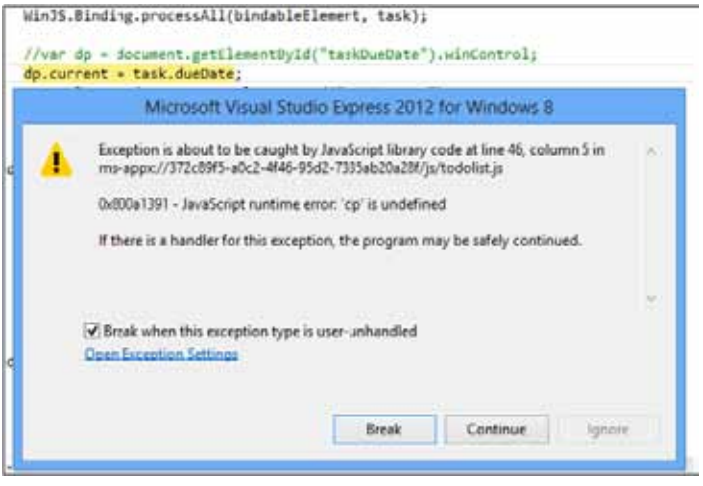

Рис. 6.13. Ошибка времени выполнения

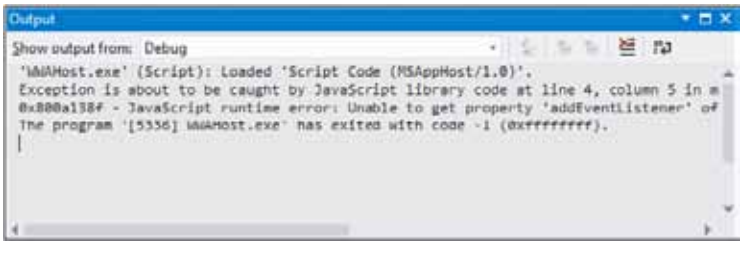

Рис. 6.14. Окно отладки

Как показано на рисунке, в окне Output (Вывод) имеется раскрывающийся список, позволяющий в ходе отладки или компоновки выбрать нужный отчет.

Проверка приемлемости вводимых данных

Завершающим шагом в создании приложения TodoList является сбор введенных данных и предоставление пользователю сводной информации, позволяющей ему перед сохранением изменений удостовериться в их правильности. Когда пользователь щелкает на кнопке Add Task (Добавить задание), программа должна проверить приемлемость данных, а затем обработать эти данные. В данном примере никакие работы с хранилищем вестись не будут, поэтому задание, определенное

пользователем, нигде сохраняется. Тем не менее вопросы проверки приемлемости данных и составления сводной информации стоит обсудить прямо сейчас.

Откройте файл todolist.js и отредактируйте функцию `TodoList.addTaskClick` следующим образом:

```
TodoList.addTaskClick = function () {
     var currentTask = TodoList.getTaskFromUI();
         if (!TodoList.validateInput(currentTask)) {
             TodoList.alert(
               "There is something wrong with the data you entered.");
             return;
         }
     TodoList.displaySummary(currentTask);
}
```

Теперь обратите внимание на функцию `TodoList.validateInput`. Эта функция получает объект `Task`, который был создан и наполнен данными, введенными через пользовательский интерфейс. К файлу todolist.js нужно также добавить следующий код:

```
TodoList.validateInput = function (task) {
     // Проверка того, что описание не оставлено пустым
         if (task.description.length == 0)
             return false;

         // Проверка того, что указана будущая дата
         if (task.dueDate <= new Date())
             return false;

         // Проверка того, что не указан положительный
         // процент выполнения задания, когда выполнение еще
         // не начиналось, или что не указан процент выполнения 100, когда
         // задание еще продолжает выполняться
         if ((task.percCompleted > 0 && task.status == "Not Started") ||
             (task.percCompleted == 100 && task.status != "Completed") ||
             (task.percCompleted != 100 && task.status == "Completed"))
             return false;
         return true;
}
```

В ответ на вопрос: «Отвечает ли переданный объект Task назначению приложения?» — функция возвращает булево значение. В конечном счете код состоит из перечня инструкций `if`, проверяющих применяемые бизнес-требования.

Составление сводки

Перед тем как получить возможность проверки задания, нужно собрать фактические данные и скопировать их в объект `Task`.

В WinJS нельзя рассчитывать на то, что система сделает за вас основную часть работы. При использовании технологий C# и XAML для создания приложения Магазина Windows систему можно заставить взять предоставляемый вам для инициализации формы объект `Task` и обновлять его при любых изменениях, вносимых пользователем

через виджеты пользовательского интерфейса. Однако эта функциональная возможность, известная как *двунаправленная привязка данных* (two-way data binding), в WinJS не поддерживается. Из-за этого вам придется добавить функцию, подобную показанной в следующем коде (ее код можно скопировать в файл todolist.js):

```
TodoList.getTaskFromUI = function () {
     var task = new Task();

     // Чтение темы задания
     var taskSubject = document.getElementById("taskSubject");
     task.description = taskSubject.value;

     // Чтение срока выполнения задания
     var taskDueDate = document.getElementById("taskDueDate").winControl;
     task.dueDate = taskDueDate.current;

     // Чтение приоритета задания
     var taskPriority = document.getElementById("taskPriority");
     task.priority = taskPriority.value;

     // Чтение статуса задания
     var taskStatus = document.getElementById("taskStatus");
     task.status = taskStatus.options[taskStatus.selectedIndex].value;

     // Чтение процента выполнения задания
     var taskPercCompleted = document.getElementById("taskPercCompleted");
     task.percCompleted = taskPercCompleted.value;
     return task;
}
```

При использовании элемента `input` его текущий контент (текст или числа) считывается с помощью свойства `value`. Для раскрывающегося списка нужно сначала извлечь свойство `selectedIndex` списка, а затем отобразить индекс на коллекцию элементов списка — эта коллекция называется `options`. И наконец, при использовании WinJS-компонента (например, компонента выбора даты) сначала с помощью свойства `winControl` нужно извлечь экземпляр этого компонента.

Теперь вы извлекли объект `Task`, переданный с данными, введенными пользователем, и этот объект может быть проверен на приемлемость данных. Если проверка приемлемости данных проходит успешно, можно продолжить и вывести сводку этих данных (или просто сохранить их где-нибудь). В завершение для вывода информации о задании на экран в отформатированном виде используется компонент `FlyOut`.

Непосредственно перед нижним колонтитулом в файл default.html нужно добавить следующую разметку, с помощью которой определяется компонент `FlyOut`, который пока остается пустым:

```
<div data-win-control="WinJS.UI.Flyout" id="flyoutSummary"></div>
```

Далее в файл todolist.js нужно добавить последний фрагмент кода — функцию `TodoList.displaySummary`. Эта функция извлечет ссылку на компонент `FlyOut`, заполнит его данными задания и покажет его пользователю.

```
TodoList.displaySummary = function (task) {
      var description = "<p><span>DESCRIPTION</span>: " +
        task.description + "<p>";
      var dueDate = "<p><span>DUE DATE</span>: " + task.dueDate + "<p>";
      var priority = "<p><span>PRIORITY</span>: " + task.priority + "<p>";
      var status = "<p><span>STATUS</span>: " + task.status + "<p>";
      var percCompleted = "<p><span>% COMPLETED</span>: " +
        task.percCompleted + "<p>";

      // Создание всей строки контента и присоединение ее к всплывающему окну
      var summary = description + dueDate + priority + status + percCompleted;
      document.getElementById("flyoutSummary").innerHTML = summary;

      // Вывод всплывающего окна
      var anchor = document.getElementById("buttonAddTask");
      var flyoutSummary = document.getElementById("flyoutSummary").winControl;
      flyoutSummary.show(anchor);
}
```

Сначала в предыдущем коде подготавливается пакет отдельных строк, соответствующих различным свойствам объекта задания, для которого нужно получить сводку. Затем из всех отдельных строк составляется сводная строка, которая программным способом прикрепляется к телу компонента `FlyOut` путем установки свойства `innerHTML` для элемента страницы с идентификатором `flyoutSummary`.

И наконец, компонент `FlyOut` выводится на экран. Учтите, что компоненту `FlyOut` нужен элемент привязки, определяющий, где появится всплывающее окно — в данном случае для привязки можно использовать кнопку Add Task (Добавить задание). Всплывающее окно со сводкой показано на рис. 6.15.

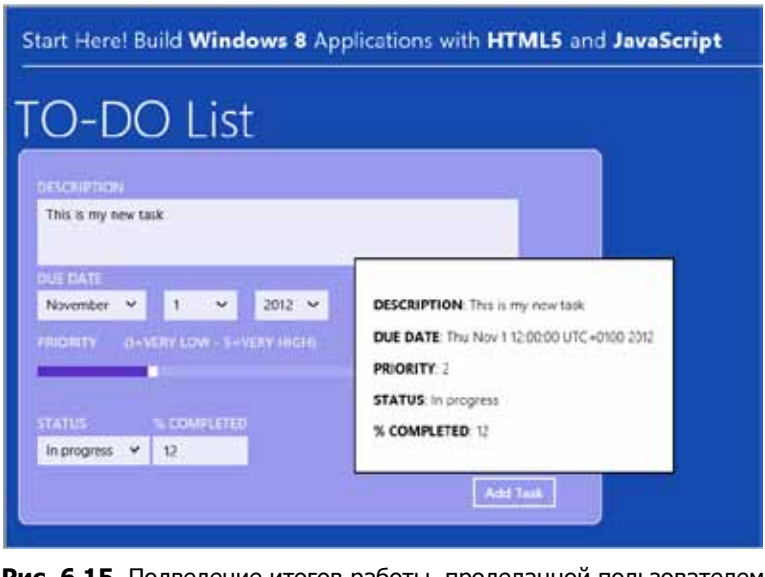

Рис. 6.15. Подведение итогов работы, проделанной пользователем

Выводы

Хотя данное упражнение нельзя признать завершенным, поскольку для сохранения задания ничего не сделано, оно продемонстрировало вам ряд аспектов разработки приложений для Windows 8. Управление пользовательским интерфейсом осуществлялось с помощью HTML-элементов и исходных компонентов Windows 8. Мы задействовали технологию привязки данных и научились структурировать JavaScript-код, необходимый для любого решения. В следующих главах мы вернемся к приложению TodoList, чтобы добавить в него ряд новых функциональных возможностей, таких как возможности сохранения и чтения заданий.

В следующей главе мы выполним несколько простых (но эффективных) упражнений, в которых будет использоваться графический и мультимедийный контент.

7 Навигация по мультимедийному контенту

Самое сложное для любого мыслителя — найти такую формулировку проблемы, которая приведет к решению.

Бертран Рассел

В этой главе исследуются возможности библиотеки WinJS (Windows 8 JavaScript) в тех ее аспектах, которые касаются вывода мультимедийного контента. Вы узнаете, как создается галерея изображений, как эти изображения увеличивать и уменьшать, как просматривать YouTube-видео с помощью приложений Магазина Windows.

Основы страничной навигации

Пока в этой книге для всех создаваемых приложений использовался шаблон Blank App (Пустое приложение). Этот шаблон хорошо подходит для приложений, состоящих из одной страницы. А что, если понадобится создать приложение, которое выводит на экран несколько страниц, заставляя пользователя переходить между ними? Но перед тем как вникать в практические детали использования изображений и видеоклипов в приложении, нам нужно познакомиться с той платформой, которая обеспечивает возможность навигации. Затем, вооружившись этими знаниями,

можно будет приступить к созданию галерей, где пользователи смогут прокручивать изображения и щелкать на них мышью для решения дополнительных задач.

Навигационная модель приложений Магазина Windows

Реализация перехода с одной страницы на другую особого труда не составляет — достаточно вызвать определенный фрагмент кода, являющийся ссылкой на другую страницу. Или, что еще проще, можно воспользоваться HTML-элементом гиперссылки.

Хотя оба этих подхода неплохо работают, следует заметить, что WinJS предлагает более богатую функциональность. Эта функциональность позволяет не только вывести другую страницу на экран, но и отследить навигационную историю пользователя, что дает вам возможность предоставить пользователю больше удобств при присоединении к существующему макету новых страниц.

Переход на другую страницу

Преимущества использования «родной» модели навигации приложений Магазина Windows хорошо иллюстрирует рис. 7.1.

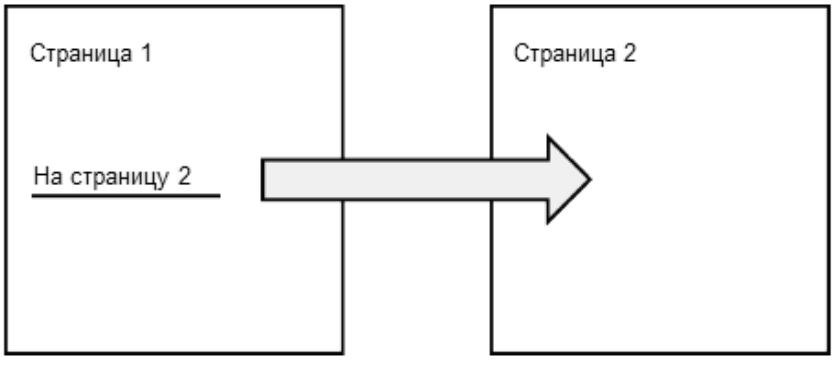

Рис. 7.1. Переход с одной страницы на другую с использованием классического HTML-элемента гиперссылки

После щелчка на ссылке (или на кнопке) пользователь видит совершенно другую страницу. При этом ответственность за создание у пользователя стойкого ощущения целостности целиком возлагается на вас. Обычно это ощущение достигается путем назначения обеим страницам одинаковой разметки, общей цветовой схемы и одинаковых шрифтов. Кроме того, пользователям нужно предоставить ссылки для возвращения на предыдущую страницу, поскольку система автоматически не предоставляет вам такую же, как у браузеров, кнопку Назад.

Вывод на экран другой страницы

Идея, заложенная в навигационную модель приложений Магазина Windows, заключается в том, что каждое приложение состоит из одной главной страницы и нескольких фрагментов. Главная страница имеет заголовок, нижний колонтитул, меню и другие стандартные элементы пользовательского интерфейса. В отличие от нее, фрагмент страницы просто предлагает новый контент, заменяющий собой контент главной страницы. Таким образом, все общие части пользовательского интерфейса при навигации остаются неизменными, создавая у пользователя ощущение изменения одного только контента, а не перехода в совершенно новое место. Навигационную модель приложений Магазина Windows иллюстрирует рис. 7.2.

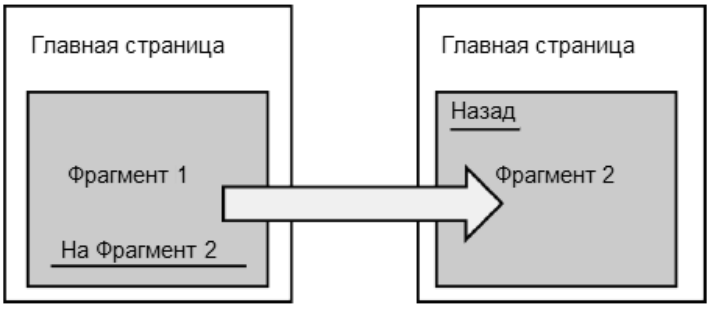

Рис. 7.2. Вывод другого контента в том же контейнере

В начальном фрагменте могут быть кнопки и другие элементы управления навигацией. Вам совершенно необязательно задействовать гиперссылки. При уходе с фрагмента страницы система сама делает всю работу, необходимую для эффективного предоставления пользователю нового контента. В то же время от вновь показанной страницы требуется лишь вывести на экран кнопку Назад. Логика обратной навигации любезно предлагается платформой WinJS.

Шаблон Navigation App

Первый пример, рассматриваемый в этой главе, станет расширением учебного приложения Магазина Windows, которое мы создадим в Microsoft Visual Studio на базе шаблона Navigation App (Приложение с навигацией). Давайте познакомимся с программной моделью.

Создание приложения с навигацией

Откройте Visual Studio и создайте новый проект на базе шаблона Navigation App. Назовите этот проект Gallery, как показано на рис. 7.3.

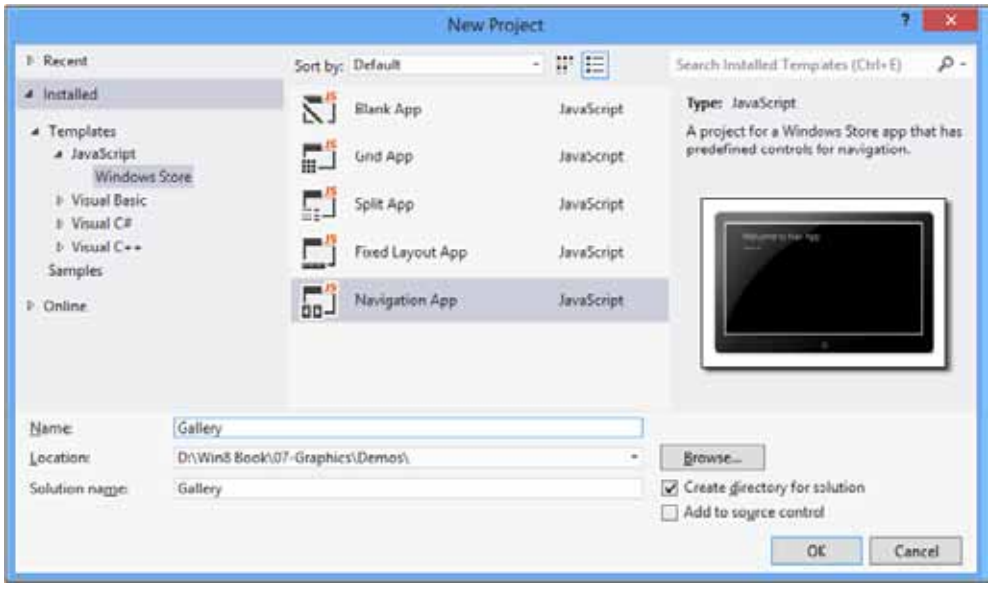

Рис. 7.3. Создание проекта на базе шаблона Navigation App

Проект содержит немного больше файлов, чем проект на базе шаблона Blank App, с которым мы работали раньше. Уже знакомая нам страница default.html играет роль главной страницы (см. рис. 7.2). Сопутствующие файлы default.css и default.js предоставляют таблицы стилей и сценарии для элементов внутри главной страницы.

На рис. 7.4 вы также видите совершенно новую папку pages. В этой папке собраны все фрагменты страниц, причем для каждой страницы выделена отдельная папка. Сначала в папке pages имеется только одна подчиненная папка по имени home, это имя является также именем единственного изначально доступного фрагмента. В папке home содержатся файлы home.html, home.css и home.js. Здесь применяется стандартная схема: CSS-файл содержит все таблицы стилей, используемые HTML-файлом, а JS-файл — все необходимые сценарии.

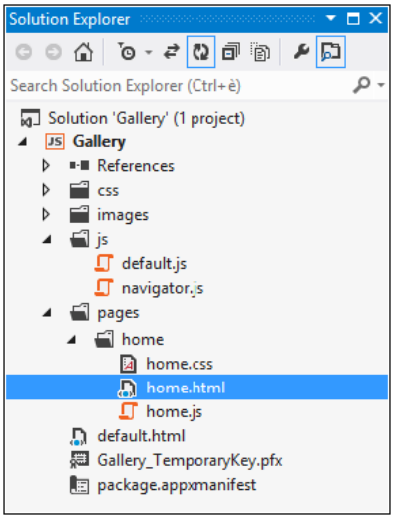

Рис. 7.4. Проект, созданный на базе шаблона Navigation App

Еще одним отличием от шаблона Blank App является наличие файла navigator.js. В этом файле содержится реализация объекта `Application.PageControlNavigator`, а сам файл можно рассматривать как расширение исходной библиотеки WinJS.

ПРИМЕЧАНИЕ

Если открыть файл default.js и сравнить его с таким же файлом, получаемым при работе с шаблоном Blank App, то и в их содержимом также будет заметна разница. В частности, в файле default.js содержится логика реализации автоматического перехода на URL-адрес, указанный в свойстве home объекта PageControlNavigator.

Задание главного экрана

В файле default.html определяется главный пользовательский интерфейс приложения. В нем содержится следующий код:

```
<div id="contenthost"
   data-win-control="Application.PageControlNavigator"
   data-win-options="{home: '/pages/home/home.html'}">
</div>
```

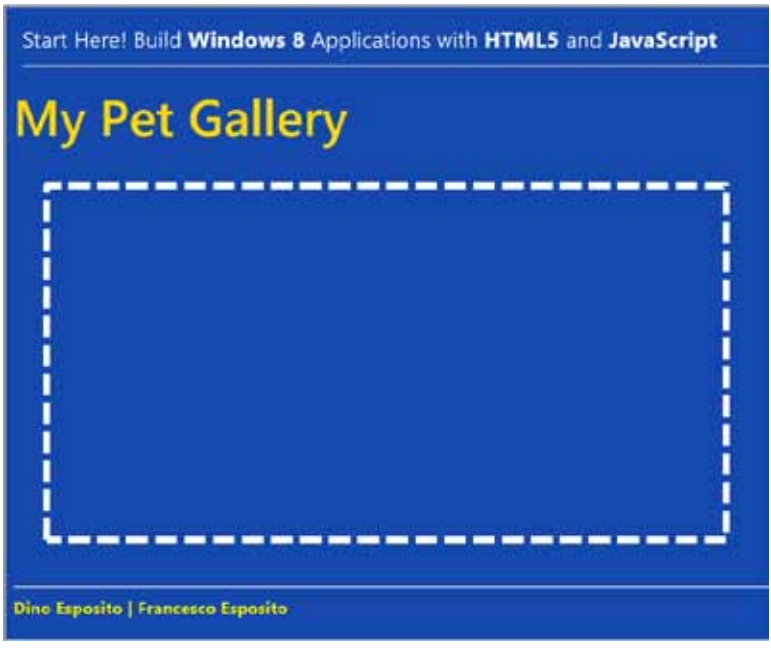

Рис. 7.5. Макет главного экрана

Этот код задает навигатор и заставляет его указывать на home.html как на поставщик фрагмента для начального экрана. Чтобы настроить главный экран, нужно внести изменения в файл default.html. Например, можно добавить в папку pages те же самые файлы header.html и footer.html, которые были созданы в предыдущей главе. Затем нужно отредактировать элемент body в файле default.html:

```
<div data-win-control="WinJS.UI.HtmlControl"
      data-win-options=»{uri:’/pages/header.html’}»></div>

 <h1 class=»title»>My Pet Gallery</h1>
 <div id=»contenthost»
   data-win-control=»Application.PageControlNavigator»
   data-win-options=»{home: ‘/pages/home/home.html’}»>
 </div>
 <div data-win-control=»WinJS.UI.HtmlControl»
     data-win-options="{uri:’/pages/footer.html’}"></div>
```

При переходе пользователя к другому фрагменту страницы меняется только контент элемента с идентификатором `contenthost`, а все остальное остается прежним. Главный экран показан на рис. 7.5, пунктирная линия обозначает область, зарезервированную для элемента с идентификатором `contenthost`. Сюда будет загружаться и визуализироваться каждая последующая страница.

После настройки данной инфраструктуры можно приступать к созданию галереи изображений.

Создание галереи изображений

Как скомпоновать галерею изображений? В основном это зависит от творческой фантазии и вкуса. В общем, галерея должна предоставлять пользователю прокручиваемый список, составленный из небольших изображений, в идеале — из уменьшенных копий (миниатюр), давая пользователю возможность щелкнуть на одном из изображений, чтобы увидеть его в большом формате и, может быть, дополнительно его увеличить. Чтобы создать галерею изображений, нужно сосредоточиться на содержимом файла home.html.

Знакомство с компонентом FlipView

WinJS поставляется с готовым к использованию компонентом `WinJS.UI.FlipView`, который хорошо подходит для создания галерей изображений. Сменяемое представление (flip view), реализуемое компонентом `FlipView`, выводится на экран как список, прокручиваемый по вертикали или по горизонтали, причем в каждый момент времени можно видеть только одну позицию в списке. Кроме того, «родной» пользовательский интерфейс компонента предлагает пользователю кнопки навигации для перехода к следующей или предыдущей позиции.

Каждая позиция может иметь любой вариант представления, имеющий смысл для приложения. Обычно наряду со сменяемым представлением (компонентом `FlipView`) вы определяете образец визуализируемой позиции списка и графический шаблон для нее.

Определение визуализируемой позиции списка

В качестве первого шага нужно определить модель для позиции списка, выводимой на экран в сменяемом представлении. Поскольку мы используем сменяемое представление для реализации галереи изображений, каждую позицию удобно представить в виде соответствующего URL-адреса физического изображения и подписи. Для этого нужно создать новый JavaScript-файл и включить его в папку js проекта. Назовите этот файл gallery.js и добавьте к нему следующий код:

```
var GalleryApp = GalleryApp || {};
 var Photo = WinJS.Class.define(function (img, title) {
   var that = {};
   that.imageUrl = img;
   that.title = title;
   return that;
 });

 GalleryApp.init = function () {
   var photos = [
     new Photo("images/data/german-sheperd.png", "German sheperd"),
     new Photo("images/data/tiger.png", "Just bigger than a cat"),
     new Photo("images/data/lion.png", "Just hairier than a cat"),
     new Photo("images/data/leopard.png", "Running as a leopard"),
     new Photo("images/data/dane.png", "Hungry from Denmark")
   ];
}
```

Объект `Photo` будет представлять собой позицию списка, выводимую на экран в сменяемом представлении (в компоненте `FlipView`). В этом простом примере вполне можно считать, что все изображения вместе с приложением включены в один пакет. В более реалистичном приложении все изображения можно получать с локального диска или, например, с сайта Flickr. Чтобы пример работал, требуется пять графических файлов с расширением .png, которые будут добавлены в папку images проекта. Чтобы отличать эти изображения от всех остальных изображений, обычно присутствующих в проекте (например, от файлов логотипов), лучше создать для них папку data, вложенную в папку images, и скопировать файлы в нее.

ПРИМЕЧАНИЕ

Не существует особых причин, по которым изображения должны быть PNG-файлами. Можно также использовать формат JPG или GIF. Но в случае PNG-файлов для них с помощью специализированных средств, например программы Paint.NET, можно задать прозрачный фон, который придаст им более привлекательный вид. Также следует заметить, что при создании галереи изображений вам может понадобиться придать единый размер всем изображениям. В нашем примере все изображения имеют размер 250 × 250 пикселов.

Создание компонента FlipView

Чтобы добавить компонент `FlipView`, откройте файл home.html и внесите в его элемент `body` следующий код:

```
<div class="fragment homepage">
   <h1>My Pet Gallery</h1>
   <div id="gallery"
     data-win-control="WinJS.UI.FlipView">
   </div>
</div>
```

Этот код просто создает новый экземпляр сменяемого представления. Следующий шаг состоит в его привязке к фотографиям. Откройте файл gallery.js и добавьте к нему следующий код:

```
var Gallery = WinJS.Class.define(function (arrayOfPhotos) {
   var that = {};
   that.photos = new WinJS.Binding.List(arrayOfPhotos);
   return that;
});
```

Затем нужно отредактировать ранее созданный метод `Gallery.init`:

```
GalleryApp.init = function () {
   var photos = [
     new Photo("images/data/german-sheperd.png", "German sheperd"),
     new Photo("images/data/tiger.png", "Just bigger than a cat"),
     new Photo("images/data/lion.png", "Just hairier than a cat"),
     new Photo("images/data/leopard.png", "Running as a leopard"),
     new Photo("images/data/dane.png", "Hungry from Denmark")
   ];
   GalleryApp.Gallery = new Gallery(photos);
   var flipView = document.getElementById("gallery").winControl;
   flipView.itemDataSource = GalleryApp.Gallery.photos.dataSource;
}
```

Теперь компонент `FlipView` привязан к массиву фотографий. Если запустить приложение сейчас, оно заработает, и сменяемое представление будет даже справляться со своей задачей. Однако такое приложение не слишком привлекательно и не слишком удобно для пользователя. Компонент `FlipView` осуществляет прокрутку связанного списка объектов `Photo`, но способен визуализировать их только в виде простого текста (рис. 7.6). Есть еще один фрагмент кода, требующий пояснения. Откройте файл home.js и добавьте вызов `GalleryApp.init` в обработчик события `ready`, чтобы компонент `FlipView` мог инициализироваться при загрузке главной страницы:

```
(function () {
   "use strict";

   WinJS.UI.Pages.define("/pages/home/home.html", {
     ready: function (element, options) {
       GalleryApp.init();
     },
```

продолжение ⇩

```
    unload: function () {
    }
  });
})();
```

Обработчик события `unload` пока пустой, но вскоре он нам тоже понадобится.

Рис. 7.6. Компонент FlipView просто показывает связанные данные

Добавление шаблона для визуализируемой позиции списка

Следующим шагом является добавление шаблона, чтобы компонент `FlipView` мог более привлекательно визуализировать связанные объекты `Photo`. Вернитесь к файлу gallery.js и добавьте к `GalleryApp.init` следующую строку:

```
flipView.itemTemplate = document.getElementById("gallery-template");
```

Нужно также в файле home.html определить шаблон `gallery-template`. Добавьте к элементу `body` в файле home.html следующий код:

```
<div id="gallery-template" data-win-control="WinJS.Binding.Template">
   <div id="template-container">
     <img class="image" alt="" src="#" data-win-bind="src: imageUrl" />
     <div class="overlay">
       <h2 data-win-bind="innerText: title"></h2>
     </div>
   </div>
</div>
```

Шаблон для визуализации позиции списка состоит из элементов `img` и `h2`. Эти элементы привязаны к свойствам объекта `Photo`. Нужно также добавить несколько стилей к файлу home.css:

```
#gallery {
    width: 600px;
    height: 350px;
    border: solid 4px #5a12f3;
    background-color: #2c668d;
 }
 .image {
    width:100%;
    border: solid 1px #fff;
 }
 #template-container {
    display: -ms-grid;
    -ms-grid-columns: 1fr;
    -ms-grid-rows: 1fr;
    width: 350px;
    height: 300px;
    background-color: #2faee6;
 }
 .overlay {
    position: relative;
    background-color: rgba(0,0,0,0.5);
    -ms-grid-row-align: end;
    height: 30px;
    padding: 10px;
    margin: 3px;
    overflow: hidden;
}
```

На этом первый этап создания учебного примера завершается: после запуска приложения вы должны увидеть то, что показано на рис. 7.7.

Рис. 7.7. Полнофункциональный компонент FlipView

Переход на подчиненную страницу

Галерея изображений не будет галереей, если не даст пользователю возможность, щелкнув на видимой позиции списка, перейти на подчиненную страницу. Именно этим мы и займемся на втором этапе упражнения.

Получение связанной позиции списка

В качестве первого шага добавьте обработчик щелчка к элементу `div`, который содержит шаблон выводимой на экран позиции списка. Нужно, как показано далее, просто добавить к элементу `div` атрибут `onclick`. Тогда при каждом щелчке пользователя на элементе `div` (или при каждом прикосновении к нему), то есть, по сути, на всей площади изображения, будет вызываться обработчик.

```
<div id="template-container" onclick="GalleryApp.showDetails()">
   ...
</div>
```

В обработчике щелчка нужно сделать две вещи. Во-первых, получить индекс выбранной позиции и по нему — связанную фотографию. Во-вторых, перейти на подчиненную страницу. Откройте файл gallery.js и добавьте следующий код:

```
GalleryApp.getCurrentPhotoIndex = function () {
   return document.getElementById("gallery").winControl.currentPage;
 };
 GalleryApp.showDetails = function () {
   var currentIndex = GalleryApp.getCurrentPhotoIndex();
   var photo = GalleryApp.Gallery.photos.getAt(currentIndex);
   WinJS.Navigation.navigate("/pages/details/details.html", photo);
};
```

Свойство `currentPage` компонента `FlipView` возвращает индекс той позиции списка, на которой был сделан щелчок. Для получения соответствующего элемента `Photo` к связанному списку применяется метод `getAt`. Связанный список — это всего лишь оболочка массива объектов, созданная только для привязки данных.

Определение подчиненной страницы

Имея в своем распоряжении выбранный объект `Photo`, можно подумать и о подчиненной странице. Назначение этой страницы заключается просто в выводе дополнительной информации о конкретном объекте `Photo`. Как показано на рис. 7.8, новый элемент Page Control (Страничный элемент управления) добавляется в папку pages/details. Перед добавлением страницы details.html нужно создать папку details.

Мастер Visual Studio добавляет к проекту три новых файла: details.html, details.css и details.js.

Элемент `body` страницы details.html должен содержать следующую разметку:

```
<div class="details fragment" type="button" />
   <header aria-label="Header content" role="banner">
```

```
    <button class="win-backbutton" aria-label="Back" disabled type="button"></
 button>
    <h1 class="titlearea win-type-ellipsis">
      <span class="pagetitle"
        data-win-bind="innerText: GalleryApp.DetailsPageModel.currentPhoto.
 title">
      </span>
    </h1>
  </header>
  <section aria-label="Main content" role="main">
  </section>
</div>
```

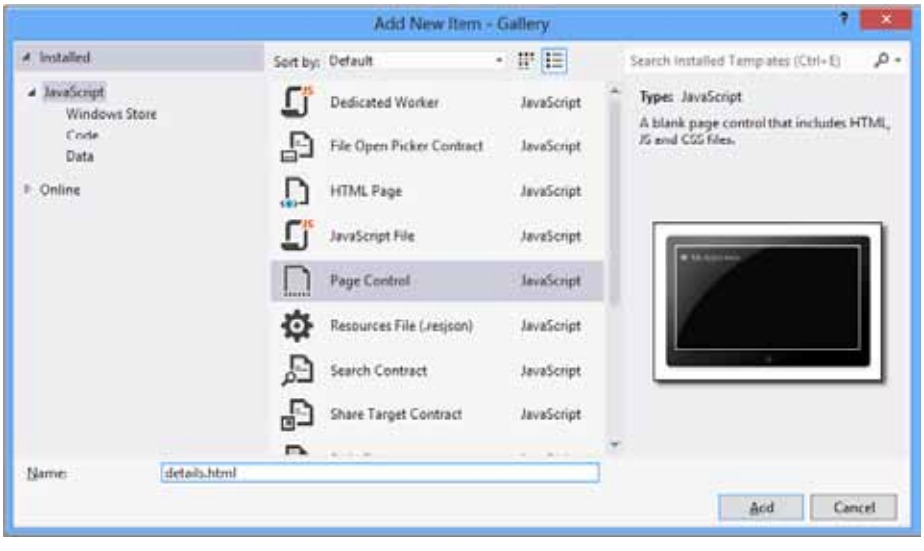

Рис. 7.8. Добавление подчиненной страницы

Наиболее важным на этой странице является элемент `header`. Он содержит кнопку Back (Назад) — с ее помощью внутренняя система навигации дает пользователю возможность вернуться к предыдущей странице. CSS-стиль `win-backbutton` обеспечивает стилизацию кнопки независимо от ее местоположения на странице.

Страница details.html привязана к данным. Ее модель представлена объектом `GalleryApp.DetailsPageModel`, который объявляется в файле gallery.js:

```
GalleryApp.DetailsPageModel = {};
```

А когда этот объект инициализируется? Это происходит в файле details.js, который нужно немного отредактировать:

```
(function () {
   "use strict";
```

продолжение ⇩

```
    WinJS.UI.Pages.define("/pages/details/details.html", {
      ready: function (element, options) {
        GalleryApp.DetailsPageModel.currentPhoto = WinJS.Navigation.state;
        WinJS.Binding.processAll();
      },
      unload: function () {
        // Следует подумать о переходе за пределы этой страницы
      }
    });
})();
```

Здесь определяется объект страницы, а также задаются два события жизненного цикла: `ready` — для события загрузки страницы и `unload` — для события ухода со страницы и ее последующей выгрузки. При обработке события `ready` завершается инициализация, для чего извлекаются и передаются данные и осуществляется их привязка к внутренней структуре данных.

Подчиненная страница может получать дополнительную информацию от вызывающей страницы. В нашем случае подчиненная страница получает информацию о выводимой на экран фотографии. Любая информация, передаваемая между страницами, извлекается через свойство `state` объекта `WinJS.Navigation`. Это дает повод рассмотреть еще один аспект навигации — обмен данными между страницами.

Обмен данными между страницами

Далее показан код, который в файле gallery.js обеспечивает переход на подчиненную страницу:

```
WinJS.Navigation.navigate("/pages/details/details.html", photo);
```

Какова роль аргумента `photo`? Она заключается в передаче параметра (или параметров) целевой странице. Это может быть любой JavaScript-объект, а его контент вы можете выбрать сами. Иными словами, можно свободно выбрать любую подходящую вам структуру этого объекта. Нужное значение извлекается целевой страницей через свойство `state` объекта `WinJS.Navigation`.

Попробуйте запустить приложение с этим кодом.

В сменяемом представлении перейдите, скажем, к третьей фотографии и щелкните на ней, чтобы перейти к подчиненной странице. На этой странице вы увидите только заголовок фотографии — все работает, как и ожидалось (рис. 7.9).

Теперь попробуйте щелкнуть на кнопке Back (Назад). Как ни странно, вы не вернетесь на ту же фотографию, с которой ушли, а попадете на первую фотографию в сменяемом представлении. Поэтому, чтобы завершить эту часть упражнения, нужно выполнить еще два дополнительных действия: найти место для хранения индекса последней просмотренной фотографии и предложить более содержательный шаблон для подчиненной страницы.

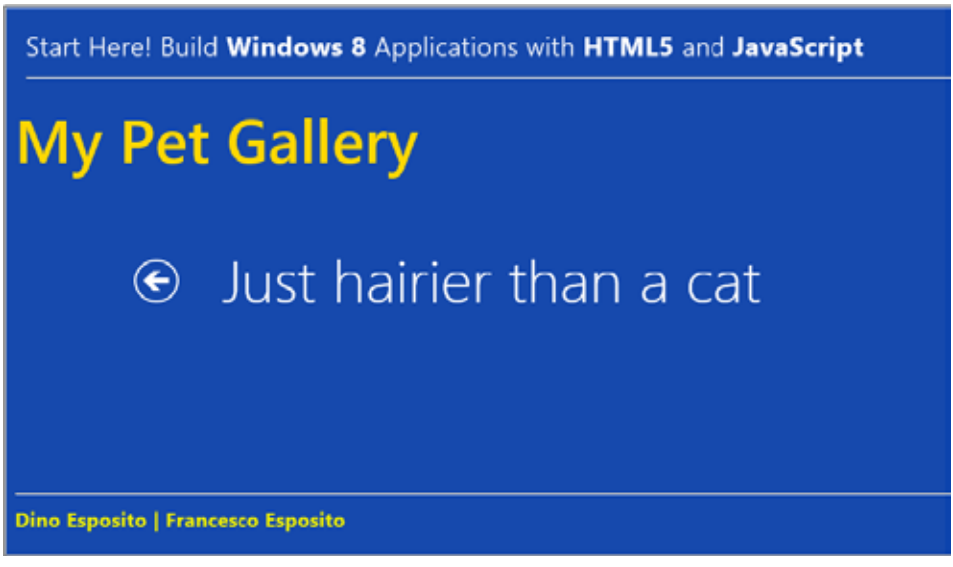

Рис. 7.9. Подчиненная страница с кнопкой Back

Сохранение информации при переходе между страницами

При переходе с одной страницы на другую контекст приложения меняется и существующие элементы управления перестают работать должным образом. Сменяемое представление (компонент `FlipView`) прекращает работу при переходе на подчиненную страницу и воссоздается при возвращении на главную страницу. Индекс фотографии, которая выводилась на экран в момент перехода, утрачивается. Чтобы исправить положение, нужно сделать три вещи:

1. Сохранить индекс текущей выведенной на экран позиции в каком-нибудь хранилище, способном «пережить» смену страниц.
2. Заставить сменяемое представление принимать еще один параметр, задающий начальную позицию.
3. Извлечь из хранилища индекс последней просматриваемой позиции и передать его в сменяемое представление, а если индекс не будет найден, заставить сменяемое представление начать с первой позиции в связанном списке позиций.

Чтобы выполнить первый шаг, нужно вернуться к файлу home.js и добавить строку к обработчику события `unload`:

```
(function () {
   "use strict";

   WinJS.UI.Pages.define("/pages/home/home.html", {
     ready: function (element, options) {
```

продолжение ⇩

```
        GalleryApp.init();
      },
      unload: function () {
        WinJS.Application.sessionState.currentPhotoIndex = GalleryApp.
        getCurrentPhotoIndex();
      }
    });
})();
```

Обработчик события `unload` отвечает за уход со страницы. Внутри этого обработчика можно сохранять индекс текущей фотографии в качестве состоянии сеанса приложения. Свойство `sessionState` объекта `WinJS.Application` является предоставляемым системой контейнером — в нем разработчики могут сохранять любую информацию, которая может использоваться для восстановления состояния приложения после его приостановки и возобновления работы. Объект `sessionState` — это обычный словарь с произвольной структурой. Иными словами, если вам нужно назначить объекту `sessionState` собственное свойство, вы его просто добавляете и используете, как это сделано со свойством `currentPhotoIndex` в предыдущем фрагменте кода.

В данном случае приложение не собирается приостанавливать свою работу, а просто переходит к следующей странице, но `sessionState` все равно работает, и, что еще важнее, вы получаете сменяемое представление, способное визуализировать себя в исходном состоянии, а также в состоянии, имевшем место на момент приостановки приложения (в случае последующего возобновления работы приложения).

ПРИМЕЧАНИЕ

Сценарий с приостановкой приложения вряд ли применим к приложениям, запущенным на ноутбуке или на настольном компьютере. Этот сценарий скорее ориентирован на приложения, работающие на мобильных устройствах. Например, приложение приостанавливается на планшетном устройстве, когда на переднем плане запускается другое приложение или когда экран блокируется.

Давайте посмотрим, как заставить сменяемое представление извлекать информацию из свойства `sessionState` и как стартовать с заданной позиции. Откройте файл gallery.js еще раз и добавьте в конце функции `GalleryApp.init` следующие строки:

```
var startIndex = WinJS.Application.sessionState.currentPhotoIndex;
flipView.currentPage = startIndex;
```

Если в свойстве `sessionState` не сохранено никакой информации, переменная `startIndex` получает значение `undefined`. Однако компонент `FlipView` достаточно разумен, чтобы понять, что значение, установленное по значению его свойства `currentPage`, не является допустимым индексом.

Масштабирование изображения

Для завершения упражнения нужно к подчиненной странице добавить еще один шаблон и еще один вариант поведения. Замысел состоит в том, чтобы вывести на экран то же изображение, на котором пользователь щелкнул в сменяемом представлении, и предоставить ему средства масштабирования этого изображения.

Определение шаблона для подчиненной страницы

Сначала нужно создать для подчиненной страницы дополнительную разметку. Откройте файл details.html и заполните элемент `section`, который уже присутствует в теле страницы, следующим кодом:

```
<section aria-label="Main content" role="main">
   <img id="zoomable-image" src="#" alt=""
     data-win-bind="src: GalleryApp.DetailsPageModel.currentPhoto.imageUrl" />
</section>
```

Теперь на подчиненной странице должно появляться то же самое изображение, которое пользователь выбирает в сменяемом представлении. А как лучше масштабировать изображение с точки зрения пользователя? Это достаточно важный вопрос.

В классических приложениях для настольных компьютеров обычный пользователь посчитает вполне естественным щелкнуть на изображении, чтобы оно увеличилось, и щелкнуть на нем же правой кнопкой мыши, чтобы оно уменьшилось. Однако приложения Магазина Windows ориентированы как на настольные компьютеры, так и на сенсорные устройства, например, планшетные компьютеры. Пользователи планшетных компьютеров, скорее всего, предпочли бы масштабировать изображение, разводя и сводя пальцы на сенсорном экране. Эта возможность в WinJS вполне реализуема, но многое оставлено на ваше усмотрение. Кроме того, это не решит проблему масштабирования для устройства, не имеющего сенсорного экрана. Избегая ненужных сложностей, можно выбрать другой подход, который хорошо работает как на сенсорных, так и на несенсорных устройствах.

ПРИМЕЧАНИЕ

Для передачи информации о прикосновении пользователя к сенсорному экрану библиотека WinJS предлагает класс GestureRecognizer. Этот класс можно использовать для определения расстояния между начальной и конечной позициями пальцев при их движении по экрану. Если расстояние имеет положительное значение, происходит увеличение масштаба, если отрицательное — уменьшение.

Добавление ползунка для выбора масштаба

Трюк, позволяющий задавать масштаб изображения (а также в целом любого масштабируемого контента) на устройствах любого типа, заключается в использовании

ползунка. Фактически, ползунком легко манипулировать как мышью, так и пальцем. Чтобы добавить ползунок к подчиненной странице, нужно в файл details.html включить следующую разметку:

```
<section aria-label="Main content" role="main">
   <input id="zoom-slider" type="range"
     min="1" max="5" value="1" step="0.1" onchange="GalleryApp.zoomImage()" />
   <bZoom level:</b
   <span id="zoom-level">1</span>
   <div id="zoomable-image-container">
     <img id="zoomable-image" src="#" alt=""
       data-win-bind="src: GalleryApp.DetailsPageModel.currentPhoto.imageUrl"
 />
   </div>
</section>
```

Ползунок назначается с помощью элемента `input` с атрибутом `type`, имеющим значение `range`. Атрибуты `min` и `max` задают диапазон значений, которые пользователь может установить с помощью этого элемента управления. Атрибут `step` показывает шаг приращения или убывания. Поскольку ползунок служит для установки масштаба, хороший диапазон формируется числами 1 и 5, а вполне комфортные ощущения дает шаг 0,1.

По умолчанию WinJS-ползунок выводит всплывающую подсказку, предлагающую пользователю визуальную обратную связь относительно устанавливаемого значения. По каким-то причинам в подсказке появляются только целые значения. Иногда это вынуждает искать какой-нибудь другой способ оповещения пользователя об установленном масштабе. В то же время вы, возможно, решите вообще отказаться от подсказки. Для этого потребуется внести небольшое изменение в файл indetails.css:

```
.details input[type=range]::-ms-tooltip {
   display: none;
}
```

Следует заметить, что лексема `.details` в выражении таблицы стилей показывает, что подсказки запрещаются только для ползунков внутри элемента, помеченного классом `details`. Класс `details` используется только для пометки тела страницы details.html. Вы можете еще раз убедиться в этом, просмотрев весь исходный код страницы details.html.

Управление изображением

В ползунке содержится обработчик события `onchange`. Этот обработчик позволяет проделать какую-нибудь работу при каждом изменении положения ползунка. В частности, может потребоваться обновить значение рядом с меткой Zoom level (Степень масштабирования) и, соответственно, изменить масштаб изображения. Нужный код добавляется к файлу gallery.js:

```
GalleryApp.zoomImage = function () {
   var container = document.getElementById("zoomable-image-container");
   var slider = document.getElementById("zoom-slider");
   var img = document.getElementById("zoomable-image");
```

```
    var label = document.getElementById("zoom-level");

    // Выполнение вычислений
    var w = container.clientWidth;
    var h = container.clientHeight;
    var zoom = slider.value;
    var offset = ((w * zoom) - w) / 2;

    // Обновление пользовательского интерфейса
    img.style.zoom = zoom;
    container.scrollLeft = offset;
    container.scrollTop = offset;
    label.innerHTML = zoom;
}
```

Масштабируемое изображение находится в контейнере `div`, стиль для которого задается ниже. Этот стиль нужно добавить в файл default.css:

```
#zoomable-image-container {
    width: 250px;
    height: 250px;
    border: solid 1px #fff;
    position: relative;
    overflow: scroll;
}
```

Значение масштаба сохраняется в переменной `zoom` и используется для вычисления величины смещения контента контейнера. Смещение представляет собой половину разницы между текущей шириной (и высотой) контейнера и действительным размером изображения с новым масштабом. В общем, требуется получить как горизонтальное, так и вертикальное смещение. Как показано в данном примере, предполагается, что изображение имеет квадратную форму, поэтому достаточно вычислить только одно смещение. Логику вычислений иллюстрирует рис. 7.10.

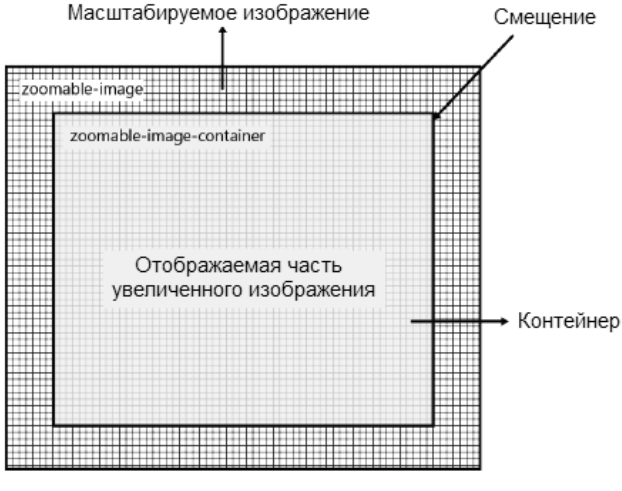

Рис. 7.10. Механизм масштабирования с использованием CSS

Конечный результат упражнения показан на рис. 7.11.

Рис. 7.11. Масштабирование изображения

Создание галереи видеоклипов

После создания галереи изображений давайте, не останавливаясь на достигнутом, попробуем создать галерею видеоклипов, используя для навигации различные элементы управления. Говоря точнее, в этом новом упражнении будет навигация, но уже внедренная в новый компонент `SemanticZoom`. В создаваемом приложении клипы будут сгруппированы по категориям, и оно будет показывать доступные клипы, как только пользователь сделает свой выбор. И наконец, при щелчке на клипе он начнет воспроизводиться.

ПРИМЕЧАНИЕ

Компонент SemanticZoom предоставляет только фиксированную форму навигации по определенному контенту в формате главный-подчиненный. В противоположность ему прикладной программный интерфейс навигации, рассмотренный в предыдущем примере, является более эффективным и универсальным, обеспечивая навигацию по любому количеству страниц и при любом количестве вложенных уровней.

Знакомство с компонентом SemanticZoom

Так как для организации перехода между главным и подчиненным представлениями мы будем опираться на встроенные возможности конкретного компонента, начинать работу с шаблона Navigation App (Приложение с навигацией) больше не нужно. Давайте создадим новый проект на базе шаблона Blank App (Пустое приложение) и назовем его `Video`. Как обычно, вам может потребоваться добавить файлы header.html и footer.html и отредактировать файл default.css, чтобы соблюсти единообразие в отношении внешнего вида приложения и ощущений пользователя при работе с ним. В частности, может потребоваться импортировать в файл default.css стили для элементов `header`, `footer` и `body`, которые использовались во всех предыдущих примерах.

ВНИМАНИЕ

Если специально не оговорено что-нибудь другое, добавление файлов header.html, footer.html, а также внесение соответствующих изменений в файл default.css потребуется для любого из последующих упражнений. Также следует заметить, что вы вряд ли испытаете какие-либо серьезные неудобства, если пропустите этот шаг, так как самое худшее, что может случиться, — это возникновение некоторых графических несоответствий!

Еще одним аспектом разработки почти всех приложений Магазина Windows, на который стоит обратить внимание в последующих упражнениях, является способ включения в приложение кода запуска. До сих пор вы добавляли в обработчик события activation в файле default.js вызов функции then. Более лаконичным, но функционально эквивалентным способом достижения того же результата является использование следующего кода:

```
app.onready = function (args) {
VideosApp.init();
};
```

Этот код помещается непосредственно в той функции, из которой обычно состоит файл default.js. Из функции onready обычно вызывается функция init для заданного объекта приложения, созданного в упражнении. Стандартный подход для всех упражнений в данной книге заключается в объединении большинства сценариев в одном JavaScript-файле с функцией init для выполнения всей инициализации.

Понятие семантического масштабирования

Несмотря на название, *семантическое масштабирование* (semantic zoom) имеет мало общего с собственно масштабированием, по крайней мере, в общепринятом значении этого понятия. Компонент `SemanticZoom` просто обеспечивает переключение между двумя разными представлениями одного и того же контента. Одно из представлений является главным, второе — подчиненным. Подчиненное представление рассматривается как увеличенное, то есть как наиболее детальное представление доступного контента. В отличие от него, главное представление является уменьшенным — в нем элементы группируются по классам, что позволяет упростить их выбор.

Подготовка почвы для семантического масштабирования

Для подготовки страницы к семантическому масштабированию требуется добавить три элемента div. Следующий код нужно ввести в элемент body страницы default.html:

```
<div id="semantic-zoom-container"
   data-win-control="WinJS.UI.SemanticZoom">
   <!-- Увеличенное представление. -->
   <div id="listview-in"
     data-win-control="WinJS.UI.ListView"></div>

   <!-- Уменьшенное представление. -->
   <div id="listview-out"
     data-win-control="WinJS.UI.ListView"></div>
</div>
```

Родительский элемент div отображается на экземпляр компонента WinJS.UI.SemanticZoom. Два дочерних элемента div отображаются на экземпляры компонентов WinJS.UI.ListView. Хорошо то, что вы отвечаете только за шаблон и контент для представлений из списка, а переключение между представлениями и пользовательский интерфейс для этого любезно предлагает компонент SemanticZoom.

Определение специальных данных для семантического масштабирования

Как обычно, к проекту добавляется новый JavaScript-файл, содержащий основную часть логики приложения. В данном случае этот файл можно назвать videos.js. Откройте этот файл и добавьте следующий код:

```
var VideosApp = VideosApp || {};

 var Clip = WinJS.Class.define(function (category, title, id) {
   var that = {};
   that.title = title;
   that.category = category;
   that.videoId = id;
   that.videoUrl = "http://www.youtube.com/embed/" + id + "?html5=1";
   that.posterUrl = "http://img.youtube.com/vi/" + id + "/1.jpg";
   return that;
});
```

Объект Clip предоставляет информацию, с которой мы будем работать в этом упражнении. К ней относится видеоклип с YouTube, у которого есть категория и название. Свойство videoId предлагает уникальный YouTube-идентификатор этого клипа. Свойство videoUrl возвращает URL-адрес, необходимый для просмотра клипа из приложения Магазина Windows, а свойство posterUrl возвращает URL-адрес, необходимый для получения постера для видео с YouTube.

Следующий этап состоит из создания привязываемой коллекции данных. Этот шаг почти не отличается от аналогичного этапа в предыдущем упражнении. Кроме того, добавьте в файл videos.js следующий код:

```
VideosApp.init = function () {
    var videos = [
      new Clip("Shots", "Top 10 Best Tennis Shots Ever", "WyJM9-7OvZo"),
      new Clip("Fun", "Very funny point", "ybsbzV7fNEo"),
      new Clip("Shots", "Best shot of 2012", "tEAkvegtPyw"),
      new Clip("Shots", "Insane passing shot", "UH5TMp_bH8k"),
      new Clip("Events", "Most important match point", "xHUwmMyRVJI")
    ];

    // Создание WinJS.Binding.List из массива
    var videosList = new WinJS.Binding.List(videos);
    var videosListGrouped = videosList.createGrouped(
      function (clip) { return clip.category; }, // группировка по этому ключу
      function (clip) { return { category: clip.category } } // данные для
                                                  // главного представления
    );
}
```

Итак, у вас имеется посвященная теннису коллекция видеоклипов, разбитая на три основные категории: `Shots`, `Fun` и `Events`. Наверное, излишне говорить о том, что эта классификация носит абсолютно произвольный характер, а названия категорий, наряду с названиями клипов, полностью отдаются на ваше усмотрение.

Группировка данных для семантического масштабирования

В приложениях Магазина Windows осуществлять привязку данных к обычным JavaScript-массивам не разрешается, сначала нужно преобразовать массивы в список привязки. Это делается посредством объекта `WinJS.Binding.List`. Именно это вы делали для заполнения компонента `FlipView` в предыдущем упражнении. Но для компонента `SemanticZoom` требуется сделать еще один шаг.

Объект списка привязки предоставляет метод `createGrouped`, посредством которого создается группируемый список привязки. Как вы уже могли видеть, метод `createGrouped` получает в качестве аргументов две функции. Первая функция просто идентифицирует ключ, по которому будут группироваться позиции списка. Вторая функция возвращает JavaScript-объект, предназначенный для визуализации каждой позиции в главном представлении. Давайте изучим код более подробно:

```
var videosListGrouped = videosList.createGrouped(
    function (clip) { return clip.category; },
    function (clip) { return { category: clip.category } }
);
```

Обе функции вызываются для каждой позиции в списке привязки. Первая функция возвращает выражение для группировки. В данном случае видеоклипы группируются по категориям. В других случаях это могут быть исходные контактные имена или комбинации нескольких свойств. Вторая функция просто возвращает обычный JavaScript-объект с информацией, которую вы намереваетесь показывать в главном представлении. Это может быть весь объект `Clip`, но в данном случае можно использовать упрощенный объект, который просто содержит категорию. То есть в главном (уменьшенном) представлении будет доступно только название категории.

Привязка данных для семантического масштабирования

Привязка данных к компоненту `SemanticZoom` и его дочерним компонентам `ListView` требует нескольких строк кода. В файле videos.js в конце функции `VideosApp.init` нужно добавить следующий код:

```
var detailView = document.getElementById("listview-in").winControl;
 detailView.itemDataSource = videosListGrouped.dataSource;
 detailView.groupDataSource = videosListGrouped.groups.dataSource;
 detailView.itemTemplate = document.getElementById("zoomed-in-template");
 detailView.groupHeaderTemplate = document.getElementById("header-template");

 var masterView = document.getElementById("listview-out").winControl;
 masterView.itemDataSource = videosListGrouped.groups.dataSource;
masterView.itemTemplate = document.getElementById("zoomed-out-template");
```

Заметьте, что для подчиненного представления данные нужно привязывать дважды. Сначала предоставляется обычный список позиций, а затем список вычисленных групп. Для получения оптимальных результатов подчиненному представлению нужен шаблон позиции списка и шаблон заголовка. Главному представлению нужен список связанных позиций и шаблон позиции.

Перед тем как можно будет впервые оценить работу, вернитесь к файлу default.html и добавьте к разметке несколько шаблонов:

```
<!-- Шаблон для заголовков групп в увеличенном подчиненном представлении -->
 <div id="header-template" data-win-control="WinJS.Binding.Template">
   <div class="header-title">
     <h1 data-win-bind="innerText: category"></h1>
   </div>
 </div>

 <!-- Шаблон для увеличенного подчиненного представления -->
 <div id="zoomed-in-template" data-win-control="WinJS.Binding.Template">
   <div class="zoomed-in-item-container">
     <img src="#" data-win-bind="src: posterUrl" />
     <div>
       <h3 data-win-bind="innerText: category"></h3>
       <h6 data-win-bind="innerText: title"></h6>
     </div>
   </div>
 </div>

 <!-- Шаблон для уменьшенного главного представления -->
 <div id="zoomed-out-template" data-win-control="WinJS.Binding.Template">
   <div class="zoomed-out-item-container">
     <h1 data-win-bind="innerText: category"></h1>
   </div>
</div>
```

Элементы в разметке богато стилизованы, поэтому на очереди дополнительная работа с файлом default.css. Откройте файл и введите следующие CSS-стили:

```
.header-title {
   width: 50px;
   height: 50px;
   padding: 8px;
 }
 .zoomed-in-item-container {
   width: 280px;
   height: 70px;
   padding: 5px;
   overflow: hidden;
   display: -ms-grid;
 }
 .zoomed-in-item-container img {
   width: 60px;
   height: 60px;
   margin: 5px;
   -ms-grid-column: 1;
 }
 .zoomed-in-item-container div {
   margin: 5px;
   -ms-grid-column: 2;
 }
 .zoomed-out-item-container {
   width: 220px;
   height: 130px;
   background-color: #31cfd4;
 }
 .zoomed-out-item-container h1 {
   padding: 10px;
   line-height: 150px;
   white-space: nowrap;
   color: #fff;
 }
 #listview-in {
   width: 650px;
   height: 300px;
   border: solid 2px #111;
 }
 #semantic-zoom-container {
   width: 600px;
   height: 350px;
   border-top: solid 2px #31cfd4;
   border-bottom: solid 2px #31cfd4;
}
```

Теперь все готово для тестирования приложения.

Использование градиентов для фона

Обратите внимание на то, что все цвета, используемые до сих пор, особенно в качестве фона, были сплошными. Поскольку в Windows 8 взят твердый курс на упрощение, сплошные цвета являются частью этой стратегии. Однако это не означает, что вы не можете использовать градиенты.

Приложения Магазина Windows полностью поддерживают стандарт CSS3, а в CSS3 имеется раздел, касающийся градиентов. Рассмотрим пример красивого (радиального) градиента, придающего фону `listview` более привлекательный вид:

```
#semantic-zoom-container {
    ...
    background: -ms-radial-gradient(
      center, ellipse cover, #c5deea 0%,#8abbd7 31%,#066dab 100%);
}
```

Синтаксис предельно прост и основан на парах цвет-процент. В данном случае градиент начинается с цвета `#c5deea`, переходит к цвету `#8abbd7`, покрывающему примерно 31 % поверхности, и заканчивается цветом `#066dab`. Если вам нравятся линейные градиенты, можно воспользоваться следующим синтаксисом:

```
background: -ms-linear-gradient(
  left, #3b679e 0%,#2b88d9 50%,#207cca 51%,#7db9e8 100%);
```

Для практических экспериментов с градиентами можно посетить сайт http://www.colorzilla.com и прямо там подобрать один из понравившихся градиентов или создать и предварительно увидеть собственные варианты.

Теперь все на месте, и результаты упражнения можно оценить (рис. 7.12).

Рис. 7.12. Компонент SemanticZoom выводит на экран свое главное представление

Заметьте, что компонент `SemanticZoom` по умолчанию выводит на экран увеличенное подчиненное представление. Чтобы заставить его показать сначала уменьшенное главное представление, в файл default.html нужно включить еще один атрибут:

```
<div id="semantic-zoom-container"
    data-win-control="WinJS.UI.SemanticZoom"
  data-win-options="{initiallyZoomedOut: true}">
```

Когда пользователь щелкает на категории (или прикасается к ней), открывается представление, показанное на рис. 7.13.

Рис. 7.13. Компонент SemanticZoom выводит на экран подчиненное представление

Чтобы вернуться к уменьшенному представлению, пользователю нужно щелкнуть на кнопке со знаком минус в правом нижнем углу экрана.

Работа с видео

Для завершения упражнения нужно дать пользователю возможность щелкнуть на элементе, рекламирующем видеоклип, чтобы начать его воспроизведение.

Обработка выбора пользователя

Компонент `ListView` поддерживает щелчки на визуализируемых позициях. Для получения уведомления о пользовательской активности нужно зарегистрировать обработчик события `iteminvoked`. Просто откройте файл default.js и добавьте в функцию `VideosApp.init` следующую строку:

```
detailView.addEventListener("iteminvoked", VideosApp.select);
```

В дополнение к этому создайте совершенно новую функцию `VideosApp.select`:

```
VideosApp.select = function (eventInfo) {
    eventInfo.detail.itemPromise.then(function (clip) {
      var player = document.getElementById("player");
      player.src = clip.data.videoUrl;
    });
}
```

Событие `iteminvoked` передает обработчикам объект `detail` со свойством `itemIndex`, которое просто возвращает начинающийся от нуля индекс позиции в списке клипов, на которой был выполнен щелчок. Однако следует заметить, что правильность индекса не гарантируется.

Иными словами, можно только предполагать, что, узнав индекс позиции щелчка, вы прекрасно справились с задачей, и остается только выбрать соответствующий объект в списке привязки. Однако в зависимости от количества определенных категорий и распределения видеоклипов по категориям полученный индекс может соответствовать, а может и не соответствовать показываемому видеоклипу. Например, можно щелкнуть с намерением воспроизвести видеоклип ABC, получив индекс три, но при этом индекс три в списке привязки может не соответствовать видеоклипу ABC.

Дело в том, что привязка элементов осуществляется в заданном порядке, но затем элементы перестраиваются на основе категорий. Получаемый индекс относится к измененному порядку следования элементов. Поэтому вам нужно заново выполнить сортировку с помощью свойства `itemPromise`:

```
eventInfo.detail.itemPromise.then(function (clip) {
    // Теперь клип соответствует позиции щелчка
}
```

Свойство `itemPromise` начинает внутренний поиск, используя индекс в качестве ключа для определения связанной позиции в списке, фактически визуализируемой в данном месте. Как только объект обнаруживается, `ListView` возвращает объект, который затем может быть обработан вашим кодом.

Воспроизведение видеоклипов с YouTube

Чтобы воспроизводить видеоклипы с YouTube на вашей странице, требуется элемент `iframe`, в который встраивается клип. Поэтому сначала нужно добавить в файл default.html следующую разметку:

```
<div id="player-container">
   <iframe id="player" height="100%" width="100%" type="text/html"></iframe>
</div>
```

В элементе `iframe` отсутствует URL-адрес видеоклипа. Этот URL-адрес предоставляется в динамическом режиме, как только пользователь щелкает на заданной позиции в представлении списка. Обычно URL-адрес клипа с YouTube выглядит так:

```
http://www.youtube.com/embed/<id>
```

Элемент `id` представляет собой идентификатор видеоклипа. Однако такой подход в приложениях Магазина Windows просто не работает. Причина в том, что по умолчанию в тело элемента `iframe` YouTube вставляет HTML-фрагмент, требующий дополнительный Flash-модуль для воспроизведения видео. Дополнительный Flash-модуль в приложениях Магазина Windows недоступен.

Чтобы исправить ситуацию для приложений Магазина Windows, нужно заставить YouTube возвращать видео в формате, совместимом с HTML5. Это делается путем использования для URL-адреса видеоклипа другого формата:

```
http://www.youtube.com/embed/<id>?html5=1
```

В результате YouTube вставляет в элемент `iframe` HTML-фрагмент, использующий HTLM5-элемент `video`, что позволяет не зависеть от дополнительного Flash-модуля. На рис. 7.14 предоставлен экран окончательной версии учебного приложения.

Рис. 7.14. Окончательная версия приложения Video Gallery

ПРИМЕЧАНИЕ

Следует иметь в виду, что в данный момент в приложениях Магазина Windows можно воспроизводить не все видеоклипы. Во-первых, владельцы видео могут заблокировать возможность внедрения видео. Во-вторых, в настоящее время YouTube не воспроизводит через HTML5-элемент video видеоклипы, содержащие анонсы или рекламу. Дополнительную информацию можно найти на сайте http://www.youtube.com/html5.

Выводы

Это главой завершается обзор уровня представления приложений Магазина Windows. Вы научились создавать и контролировать специализированные представления, включая сменяемые представления, представления списка и представления формата главное-подчиненное (упоминаемые в контексте так называемого семантического масштабирования). Мы также получили опыт работы с системой навигации и узнали о том, что требуется для перехода на другую страницу и передачи на эту страницу информации. В данной главе мы также коснулись вопроса сохранения данных состояния приложения в ходе сеанса работы с ним.

Затем мы выполнили два необычных упражнения и создали две интерактивные галереи для изображений и видеоклипов. В этих упражнениях мы работали с фиксированными коллекциями изображений и видеоклипов, изучая основы создания галерей и взаимодействия с ними. После обучения базовым приемам работы с файлами и данными вы будете готовы к созданию настоящих приложений для Windows 8.

Следующая глава снова посвящена визуальным аспектам приложения. Но главным в ней будут не новые виджеты и компоненты, а состояния приложения и связанные с ними режимы вывода информации.

8 Состояния приложений для Windows 8

Цель без плана — это просто пожелание.

Антуан де Сент-Экзюпери

Пару десятилетий назад идея выпуска операционной системы на базе окон, в которой бок о бок могли бы уживаться сразу несколько приложений, имела достаточно революционный характер. До выхода Microsoft Windows (и других подобных операционных систем) пользователь мог одновременно работать только с одним приложением. Активное приложение полностью брало на себя управление машиной и ее вычислительными ресурсами, заполняя своим контентом весь экран.

В последнее время идея одного приложения, работающего на переднем плане, была реанимирована для таких мобильных операционных систем, как iOS, Windows Phone и Android. После многих лет существования оконных операционных систем, в которых нескольких приложений могли параллельно работать в отдельных окнах, модель единственного выполняющегося приложения стала шоком для многих пользователей.

А что насчет Microsoft Windows 8?

Если Windows 8 работает в режиме настольного компьютера, тогда это та же самая привычная нам система Windows с многочисленными перекрывающими друг друга окнами, которые можно открывать в любое время. Однако если вы запустите приложение Магазина Windows, то обнаружите, что каждое приложение, *как правило*, заполняет собой весь экран, не давая вам возможности взаимодействовать с другими

приложениями, не переключившись на них и не позволив им, в свою очередь, заполнить весь экран. Точнее говоря, приложения Магазина Windows могут находиться в нескольких состояниях, в том числе в полноэкранном (как с книжной, так и с альбомной ориентацией), а также в состояниях заполнения и закрепления. Если приложение Магазина Windows не запущено в каком-нибудь из полноэкранных состояний, оно делит доступный экран со вторым приложением, и тогда говорят, что одно из приложений выполняется в *состоянии закрепления* (snapped state), второе — в *состоянии заполнения* (filled state).

Но что это означает для вас?

В идеале приложение должно уметь в определенной степени приспосабливаться, чтобы его состояние сохранялось, а пользовательский интерфейс адаптировался к меньшему пространству. Основной темой данной главы являются визуальные состояния закрепления и заполнения.

Состояния приложений Магазина Windows

Приложения Магазина Windows должны предоставлять пользователям возможность полноэкранного и многоканального восприятия информации. С одной стороны, руководства по дизайну рекомендуют сохранять достаточную простоту пользовательского интерфейса и не заполнять экран слишком большим количеством объектов. С другой стороны, по мере распространения экранов с высоким разрешением возникает риск того, что на большом экране будет совсем немного контента.

Поскольку изначально операционная система Windows 8 была ориентирована на устройства разных размеров, тема адаптации пользовательского интерфейса (и, следовательно, пользовательского восприятия) к различным экранам является в целом актуальной.

Windows 8 предлагает несколько визуальных состояний для обычных приложений и приложений Магазина Windows с получением соответствующего уведомления при изменении визуального состояния. Все, что происходит потом, зависит от самого приложения. Давайте сначала поговорим о предопределенных визуальных состояниях.

Полноэкранные визуальные состояния

Приложение, запущенное в полноэкранном режиме, занимает весь экран. Если при запуске приложения не указано ничего иного, это визуальное состояние предлагается по умолчанию для любого приложения Магазина Windows. Точнее говоря, в Windows 8 определено два полноэкранных режима, по одному для каждого варианта ориентации устройства.

Альбомная ориентация

Альбомная ориентация служит признаком горизонтального расположения устройства, когда ширина экрана больше его высоты. Этот режим может быть выявлен программно путем исследования значения, возвращаемого объектом `Windows.UI.ViewManagement.ApplicationViewState`. Как это делается, показано в следующем фрагменте кода:

```
// Получение текущего визуального состояния
 var currentState = Windows.UI.ViewManagement.ApplicationView.value;

 // Проверка того, что приложение выполняется в полноэкранном
 // режиме с альбомной ориентацией
 if (currentState === Windows.UI.ViewManagement.ApplicationViewState.
                                                    fullScreenLandscape) {
 ...
}
```

Операционная система обнаруживает поворот устройства и автоматически перестраивает пользовательский интерфейс, уведомляя приложение о любом изменении.

Книжная ориентация

Книжная ориентация свидетельствует о том, что высота доступного экрана больше его ширины. В этом случае предполагается, что вертикальный вывод контента предпочтительнее для приложения, поскольку при этом пользователь получает больше пространства для заполнения и ему проще прокручивать данные вниз, чем сдвигать их влево. Обнаружить книжную ориентацию можно с помощью следующего фрагмента кода:

```
// Получение текущего визуального состояния
 var currentState = Windows.UI.ViewManagement.ApplicationView.value;

 // Проверка того, что приложение выполняется в полноэкранном
 // режиме с книжной ориентацией
 if (currentState === Windows.UI.ViewManagement.ApplicationViewState.
                                                    fullScreenPortrait) {
 ...
}
```

Как и в предыдущем случае, операционная система обнаруживает поворот устройства и автоматически предписывает приложению перерисовать пользовательский интерфейс.

ВНИМАНИЕ

Получается, что вращение устройства в полноэкранном режиме может потребовать определенной адаптации всего пользовательского интерфейса приложения. Зачастую контент, изначально разработанный с расчетом на его расширение по горизонтали, нуждается в реорганизации, чтобы так же хорошо выглядеть на устройстве, удерживаемом в книжной ориентации. Операционная система этого за вас не сделает, ограничившись предоставлением приложению возможности узнать об изменении условий просмотра и предписанием перерисовать интерфейс.

Закрепление приложений

Пользователь может перевести в состояние закрепления одно из приложений, которое в этот момент выполняется в фоновом режиме. Для этого нужно вывести список переключений на левой стороне экрана устройства и щелкнуть правой кнопкой мыши на плитке выбранного приложения. Как показано на рис. 8.1, вы можете выбрать, где закрепить приложение, у левого (Snap left) или у правого (Snap right) края экрана.

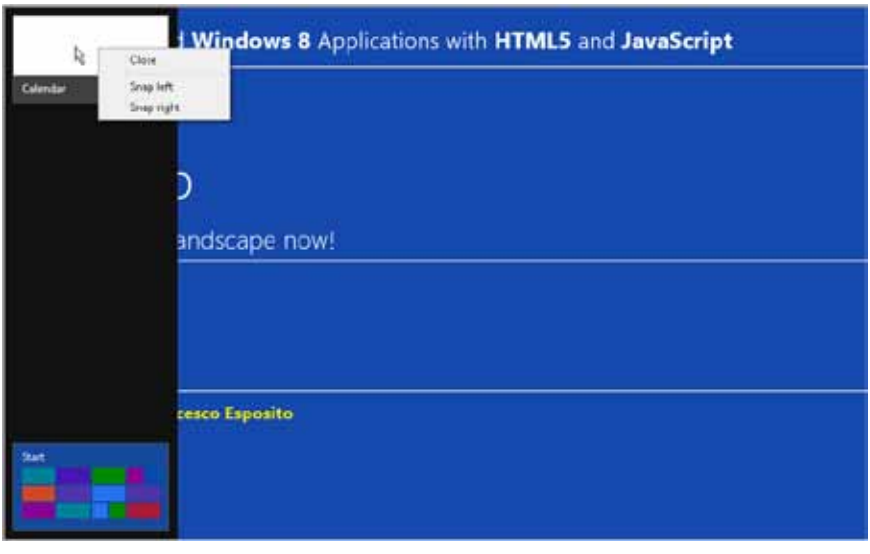

Рис. 8.1. Перевод приложения, выполняемого в фоновом режиме, в состояние закрепления

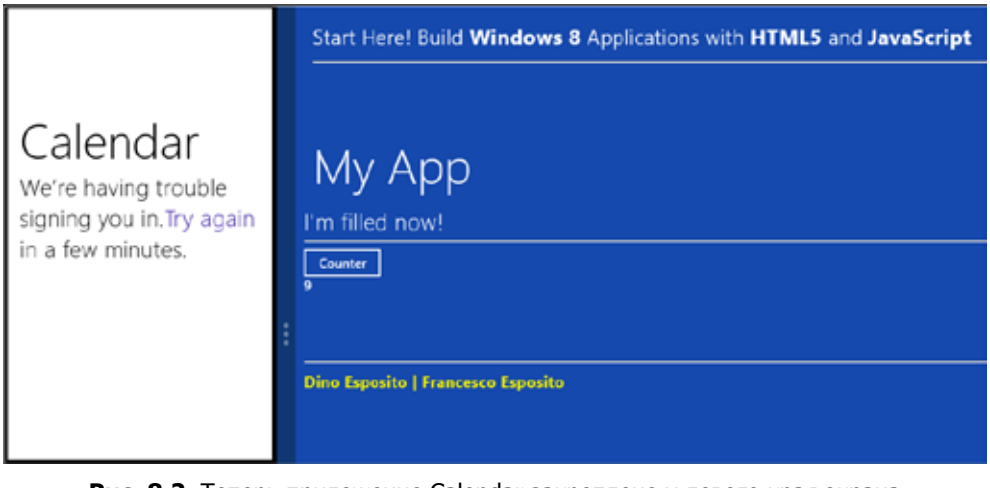

Рис. 8.2. Теперь приложение Calendar закреплено у левого края экрана

Результат показан на рис. 8.2. После включения режима закрепления визуальное состояние приложения соответствующим образом меняется, а другое приложение, ранее работавшее в полноэкранном режиме, заполняет всю оставшуюся часть экрана, переходя в визуальное состояние *заполнения*.

Визуальное состояние закрепления

В состоянии закрепления приложение меняет свои размеры, получая сегмент экрана размером 320 пикселов в ширину и весь экран в высоту. Как показано на рис. 8.2, весь остальной экран занимает приложение, которое до закрепления другого приложения выполнялось на переднем плане. Таким образом, при закреплении у пользователя на переднем плане могут находиться два приложения, что позволяет работать с ними одновременно.

ВНИМАНИЕ

Относительно закрепления, поддерживаемого в Windows 8, нужно отметить два ключевых момента. Во-первых, закрепление возможно, только если горизонтальное разрешение экрана составляет как минимум 1366 пикселов. Пользователь, попытавшийся закрепить приложение на экране с меньшим разрешением, просто увидит контекстное меню, показанное на рис. 8.1, в котором будет единственный пункт — Close (Закрыть). Во-вторых, закрепление возможно только при альбомной ориентации устройства. При наличии закрепленного приложения и последующей смены ориентации устройства на книжную эффект утрачивается, то есть незакрепленное приложение с книжной ориентацией снова получает в свое распоряжение весь экран. Тем не менее режим закрепления при этом не отключается, и визуальное состояние закрепления восстанавливается сразу же после того, как пользователь возвращает устройству альбомную ориентацию.

Кроме того, нужно иметь в виду, что пользователям не разрешается произвольно менять размер области закрепления. Границу, которая на рис. 8.2 разделяет приложения в состояниях закрепления и заполнения, нельзя перемещать по вашему желанию. Если пользователь попытается сместить эту разделительную панель вправо, он может получить один из следующих эффектов: во-первых, два приложения могут поменяться своими визуальными состояниями, во-вторых, приложение в состоянии закрепления может перейти в полноэкранный режим, заставив второе приложение переключиться в фоновый режим. В первом случае результатом станет то, что приложение Calendar перейдет в состояние заполнения, а второе приложение окажется закрепленным у правого края экрана.

Визуальное состояние заполнения

Когда включен режим закрепления, горизонтальное пространство экрана делится между двумя приложениями, находящимися в состояниях закрепления и заполнения (рис. 8.3).

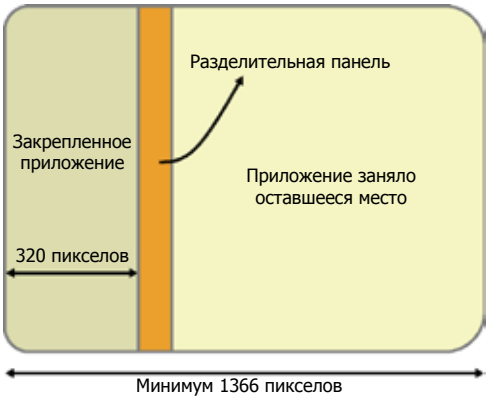

Рис. 8.3. Основные признаки приложений в состояниях закрепления и заполнения

В течение своего жизненного цикла любое приложение может иметь оба визуальных состояния. Решение о режиме вывода на экран принимает пользователь, а не приложение. Со стороны приложения поддержка визуальных состояний (включая состояния закрепление и заполнения) означает готовность визуализировать любой контент на экране любых размеров.

Считается, что поведение приложения Магазина Windows не нужно менять при смене его визуального состояния с полноэкранного на любое другое. На этот счет имеются весьма конкретные рекомендации. То есть вполне *разумно* ожидать очень незначительных изменений в том, что касается поведения приложения и его пользовательского интерфейса при работе в разных режимах (полноэкранном с альбомной или портретной ориентацией либо в режиме заполнения). В то же время не все приложения могут меняться в размере, подстраиваясь под некий осколок исходного экрана, и при этом сохранять стопроцентную функциональность.

Вскоре вы узнаете, как обнаружить изменение визуального состояния приложения Магазина Windows и что это может означать для вашего приложения с точки зрения поддержки различных вариантов экранного разрешения. Но перед тем как двигаться дальше, взгляните на рис. 8.4.

Вместо копии экрана на этом рисунке показана схема, иллюстрирующая сложности, с которыми сталкиваются некоторые приложения при адаптации к различным экранным разрешениям.

После закрепления приложению остается всего 320 пикселов в ширину для визуализации своего пользовательского интерфейса. В большинстве других случаев ширина как минимум втрое больше.

Золотое правило приложений Магазина Windows можно вкратце свести к следующему: «Будьте готовы к тому, чтобы делать что-то серьезное на площади размером примерно 320 × 760 пикселов».

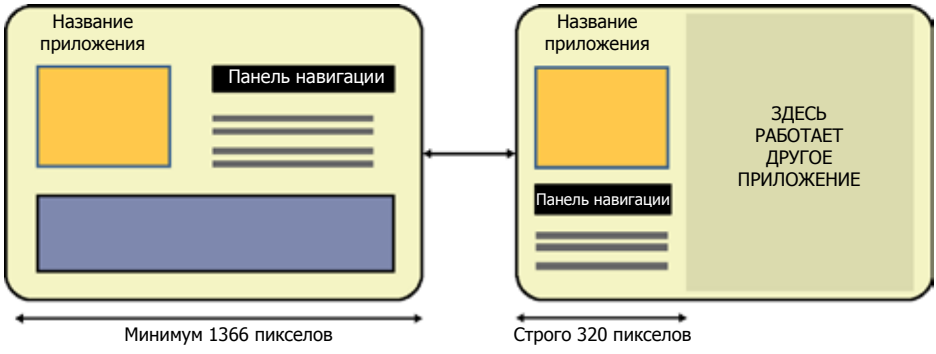

Рис. 8.4. Как адаптировать пользовательский интерфейс при переходе от полноэкранного представления к режиму закрепления?

Наделение приложения способностью реагировать на изменение состояния

Теперь посмотрим, что должно сделать приложение, чтобы обнаружить и, возможно, обработать изменение визуального состояния. Создайте новое приложение Магазина Windows на базе шаблона Blank App (Пустое приложение). Это приложение можно назвать SnapMe. Затем внесите в проект все начальные изменения, которые делались во всех предыдущих упражнениях. Сюда относятся, к примеру, добавление заголовка и колонтитула, а также ряда стилей к файлу default.css, внесение необходимых изменений в связанный с приложением файл сценариев. Назовите этот новый файл сценариев SnapMeApp.js.

Новые нюансы в разработке приложений

В предыдущих главах мы не уделяли особого внимания состоянию приложений. Фактически все упражнения касались приложений, не сохраняющих своего состояния. Почти у каждого приложения имеется состояние, которое обновляется по мере работы пользователя с этим приложением. При приостановке приложения состояние должно сохраняться в постоянном хранилище, тогда можно будет легко восстановить это состояние, если выполнение приложения возобновится или если приложение будет перезапущено, и только тогда у пользователя появится ощущение целостности.

Задавшись этой целью, нужно добавить к только что созданному файлу SnapMeApp.js следующий код:

```
var SnapMeApp = SnapMeApp || {};

 var SnapMeState = WinJS.Class.define(function () {
   var that = {};
```

продолжение ⇩

```
    that.currentViewState = SnapMeApp.getViewStateForDisplay();
    that.total = 0;
    return that;
  });

  SnapMeApp.init = function () {
    SnapMeApp.Current = new SnapMeState();
    var buttonCounter = document.getElementById("buttonCounter");
    buttonCounter.addEventListener("click", SnapMeApp.add);
    SnapMeApp.refresh();
}
```

Создаваемое нами учебное приложение будет содержать только кнопку, по щелчку на которой увеличивается значение счетчика. Класс `SnapMeState` описывает состояние приложения. В свойстве `total` этого класса ведется подсчет щелчков. Свойство `currentState` содержит сообщение о текущем визуальном состоянии приложения.

В функции `SnapMeApp.init` происходит инициализация состояния приложения и регистрация обработчика события `click` кнопки. Также регистрируется функция `SnapMeApp.init`, которая вызывается при готовности приложения. В файле default.js должен быть следующий код:

```
app.onready = function (args) {
    SnapMeApp.init();
};
```

В тело файла страницы default.html, расположенной между страницами заголовка и нижнего колонтитула, нужно поместить такую разметку:

```
<h1>My App</h1>
  <h2 id="currentViewState"></h2>
  <hr />
  <button id="buttonCounter">Counter</button>
<h3 id="total"></h3>
```

Помимо сохранения состояния еще одной практической концепцией является разделение состояния, поведения и логики обновления пользовательского интерфейса. Чтобы выразить поведение, связанное со щелчком на кнопке, нужно в нижнюю часть файла SnapMeApp.js добавить следующий код:

```
SnapMeApp.add = function () {
    // Это часть поведения приложения
    SnapMeApp.Current.total++;

    // Этот код обновляет пользовательский интерфейс
    SnapMeApp.refresh();
}
```

И наконец, для обновления пользовательского интерфейса нужен следующий код:

```
SnapMeApp.refresh = function () {
    // Обновление метки текущим визуальным состоянием
    var stateElem = document.getElementById("currentViewState");
```

```
    stateElem.innerHTML = SnapMeApp.Current.currentViewState;

    // Обновление метки текущим количеством щелчков
    var totalElem = document.getElementById("total");
    totalElem.innerHTML = SnapMeApp.Current.total;
}
```

При наличии функции `SnapMeApp.refresh` вам не нужно извлекать информацию из элементов управления перед обновлением пользовательского интерфейса. Точно так же вам не нужно обновлять пользовательский интерфейс напрямую из кода обработки событий, касающихся визуальных элементов. В результате код в целом становится понятнее и проще в разработке и сопровождении.

Обнаружение изменений в визуальном состоянии

Следующий шаг заключается в добавлении такого кода к приложению SnapMe, который даст ему возможность обнаруживать изменения в визуальном состоянии и соответствующим образом реагировать на это. В результате вам останется только записать новое визуальное состояние и показать его пользователю в метке `currentViewState`.

Любое приложение Магазина Windows, написанное с использованием JavaScript и HTML, получает уведомление об изменении своего визуального состояния через событие `onresize` объекта `window`. Это событие возникает, когда окно, в котором выполняется приложение, меняет свой размер. В Windows 8 это может произойти только при изменении ориентации устройства либо при закреплении или откреплении приложения. Для получения события `onresize` нужно зарегистрировать обработчик. Лучше всего это сделать в функции `SnapMeApp.init`. Окончательная версия кода функции выглядит так:

```
SnapMeApp.init = function () {
    window.onresize = addEventListener('resize', SnapMeApp.onResize, false);

    SnapMeApp.Current = new SnapMeState();
    var buttonCounter = document.getElementById("buttonCounter");
    buttonCounter.addEventListener("click", SnapMeApp.add);

    SnapMeApp.refresh();
}
```

Далее нужно добавить в конец файла SnapMeApp.js новую функцию `SnapMeApp.onResize`:

```
SnapMeApp.onResize = function (e) {
    // Определение текущего визуального состояния и сохранение
    // его в виде строки
    SnapMeApp.Current.currentViewState = SnapMeApp.getViewStateForDisplay();

    // Обновление пользовательского интерфейса
    SnapMeApp.refresh();
}
```

Как уже отмечалось, текущее визуальное состояние возвращается в виде целого числа с помощью предоставляемого системой глобального объекта — перечисления `Windows.UI.ViewManagement.ApplicationViewState`. Это перечисление вычисляет четыре возможных значения — по одному для каждого из возможных визуальных состояний. Выяснение конкретного визуального состояния упрощается для разработчика благодаря наличию четырех готовых к использованию констант:

```
Windows.UI.ViewManagement.ApplicationViewState.snapped
  Windows.UI.ViewManagement.ApplicationViewState.filled
  Windows.UI.ViewManagement.ApplicationViewState.fullScreenLandscape
Windows.UI.ViewManagement.ApplicationViewState.fullScreenPortrait
```

Следующий фрагмент из файла SnapMeApp.js превращает код, обозначающий визуальное состояние, в читабельный текст:

```
SnapMeApp.getViewStateForDisplay = function () {
    var viewState = Windows.UI.ViewManagement.ApplicationView.value;
    switch (viewState) {
      case Windows.UI.ViewManagement.ApplicationViewState.snapped:
        return "I'm snapped now!";
      case Windows.UI.ViewManagement.ApplicationViewState.filled:
        return "I'm filled now!";
      case Windows.UI.ViewManagement.ApplicationViewState.fullScreenLandscape:
        return "I'm full screen landscape now!";
      case Windows.UI.ViewManagement.ApplicationViewState.fullScreenPortrait:
        return "I'm full screen portrait now!";
    }
}
```

Теперь все готово к компиляции учебного приложения. Результат показан на рис. 8.5. Судя по выведенному на экран сообщению: I'm full screen landscape now!, — приложение изначально запускается в полноэкранном режиме с альбомной ориентацией. Обратите внимание на то, что, если приложение будет работать на устройстве с книжной ориентацией, вы получите другое описание визуального состояния: I'm full screen portrait now!.

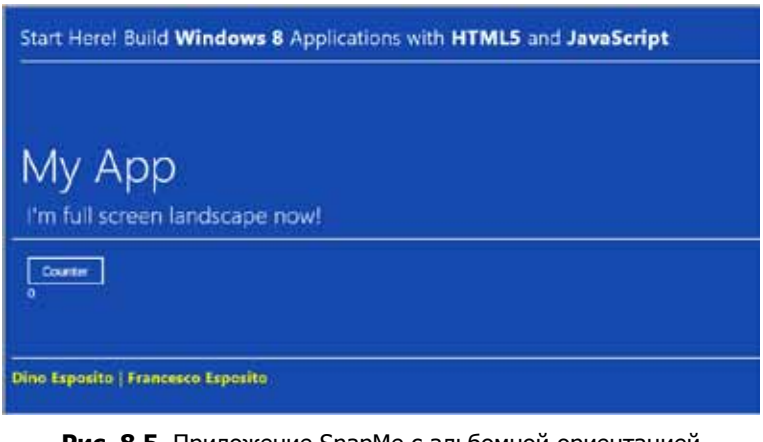

Рис. 8.5. Приложение SnapMe с альбомной ориентацией

Интересно отметить, что для экспериментов с ориентацией в Microsoft Visual Studio есть имитатор со встроенными функциями. На рис. 8.6 показано, как сменить ориентацию в имитаторе, чтобы проверить, правильно ли приложение SnapMe реагирует на книжную ориентацию.

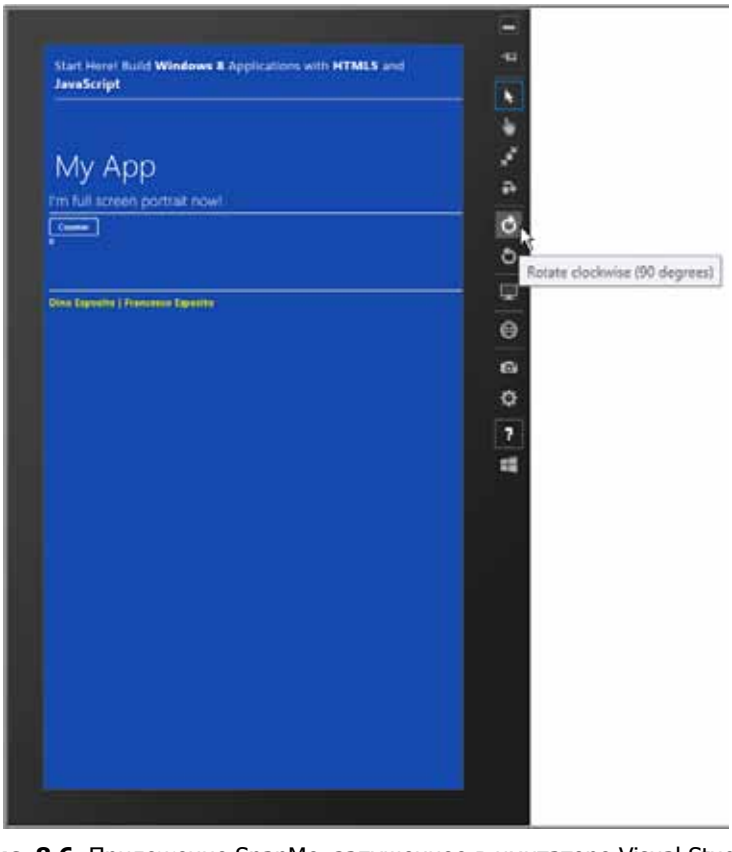

Рис. 8.6. Приложение SnapMe, запущенное в имитаторе Visual Studio

И наконец, на рис. 8.7 показан пользовательский интерфейс учебного приложения, когда оно закреплено у левого края экрана.

Когда пользователь щелкает на кнопке, свойство `total` объекта состояния приложения обновляется и выводится на экран через пользовательский интерфейс. А что происходит с состоянием, когда приложение закрепляется, открепляется или когда изменяется ориентация экрана?

События изменения ориентации и представления влияют на состояния приложения, если приложение находится на переднем плане независимо от режима просмотра (полноэкранный режим либо режим закрепления или заполнения). В фоновом режиме приложение этих событий вообще не получает.

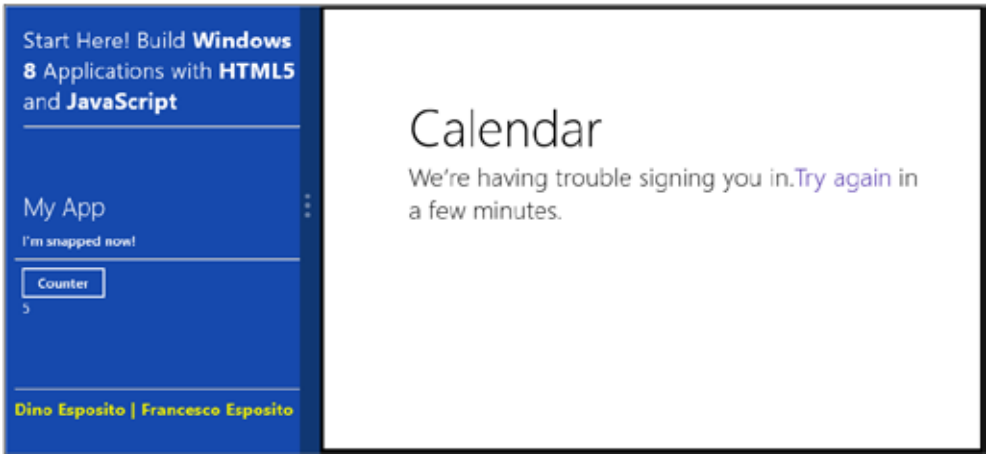

Рис. 8.7. Приложение SnapMe сейчас закреплено

Адаптация контента приложения

До сих пор мы еще по-настоящему не сталкивались со сложностями поддержания приложением нескольких вариантов экранного разрешения. Чтобы разобраться с этой проблемой, внесите в файл default.html следующие изменения:

```
<div style="width: 500px; background-color: red;">
    <h1>My App</h1>
    <h2 style="text-align: right" id="currentViewState"></h2>
    <hr />
    <button id="buttonCounter">Counter</button>
    <h3 id="total"></h3>
</div>
```

Тем самым главной странице придается немного другое стилевое оформление. Теперь весь пользовательский интерфейс страницы (кроме заголовка и нижнего колонтитула) заключен в пространство шириной 500 пикселов и выделен красным цветом. Кроме того, сообщение о текущем визуальном состоянии теперь выровнено по правому краю контейнера, выделенного для пользовательского интерфейса. Ширина для этого прямоугольного контейнера (500 пикселов) взята не случайно: требуется значение, существенно превышающее 320 пикселов. Следует вспомнить, что 320 пикселов — это ширина области закрепления.

Приложение не испытывает никаких конкретных проблем при работе в режиме заполнения с альбомной или книжной ориентацией. Однако это скорее исключение, чем правило. В режиме заполнения с альбомной или книжной ориентацией приложение с различными экранными разрешениями и, что важнее, при различных соотношениях сторон по-прежнему работает правильно, но, если закрепить приложение SnapMe у края экрана, вы увидите то, что показано на рис. 8.8.

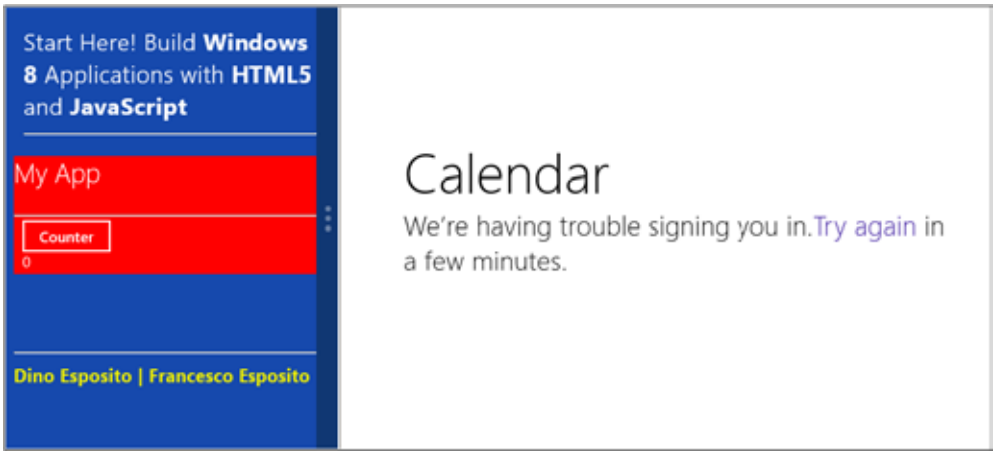

Рис. 8.8. В режиме закрепления часть контента оказывается обрезанной

Теперь надпись I'm snapped now! больше не видна. Причина в том, что окно вынуждено принять фиксированную ширину, и текст, который после последних изменений выровнен по правому краю, в доступное пространство больше не помещается.

Этот пример иллюстрирует ряд проблем визуализации, полностью справиться с которыми можно, только снабдив приложение адаптивным макетом.

Переход к адаптивному макету

Когда дело доходит до режимов закрепления и заполнения, главным фактором становится то, что пользователь всегда может закрепить любое приложение. Ваше приложение может адаптироваться к состоянию закрепления, но не может воспрепятствовать закреплению. Если приложение разрабатывалось без учета возможности закрепления, то при принудительном изменении размера занимаемого им пространства часть контента может оказаться обрезанной, как было показано на рис. 8.8.

Основные принципы организации режимов закрепления и заполнения

Все приложения Магазина Windows должны следовать ряду рекомендаций, касающихся представлений в режимах закрепления и заполнения. Но на самом деле могут возникать такие сценарии и ситуации, в которых полностью следовать этим рекомендациям просто невозможно. И тем не менее нужно приложить все усилия,

чтобы разработать такой макет, который мог бы хорошо адаптироваться к любому размеру. Рассмотрим принципы, лежащие в основе организации представления в режиме закрепления в Windows 8.

Отслеживайте изменение представления

Любое приложение Магазина Windows всегда должно регистрировать обработчик события изменения размеров окна. Это позволит обеспечить соответствующую реакцию на любое существенное изменение представления. В предыдущем упражнении мы проверили этот принцип на практике.

Поддерживайте одинаковую функциональность во всех состояниях

В идеале должно существовать только *одно* приложение, предлагающее некое универсальное поведение для всех возможных визуальных состояний. В любой ситуации приложение должно демонстрировать стопроцентную функциональность независимо от своего визуального состояния, будь то закрепление или заполнение, альбомная или книжная ориентация.

Однако на практике этот принцип нельзя возвести в ранг закона, скорее, он требует гибкого толкования. Что делать, если ваше приложение должно выводить на экран большую форму для ввода данных? Что, если приложению приходится выводить на экран большие объекты с данными, например видеосюжеты или фотографии? И видео и фото до некоторой степени можно сжать, но их качество при этом может соответствовать, а может и не соответствовать назначению приложения.

Аккуратно открепляйте приложение

Если обеспечить одинаковую функциональность нельзя, нужно предусмотреть возможность программного открепления приложения, когда пользователь выбирает одну из критически важных функций. Рассмотрим в качестве примера приложение со списком неотложных дел.

Создавать новое задание удобнее, наверное, располагая достаточно большой областью экрана. Но в то же время для просмотра заданий вполне достаточно пространства экрана шириной 320 пикселов, что соответствует представлению в режиме закрепления. Поэтому, когда пользователь щелкает на кнопке Add new task (Добавить новое задание), нужно программным способом открепить приложение. Чтобы попрактиковаться в откреплении приложений, добавим к функции `SnapMeApp.add` в файле SnapMeAdd.js следующий код:

```
SnapMeApp.add = function () {
    SnapMeApp.Current.total++;
    var viewState = Windows.UI.ViewManagement.ApplicationViewState;
    if (SnapMeApp.Current.total >== 10 &&
```

```
      viewState === Windows.UI.ViewManagement.ApplicationViewState.snapped) {
      Windows.UI.ViewManagement.ApplicationView.tryUnsnap();
    }
    SnapMeApp.refresh();
}
```

В этом примере, когда счетчик доходит до порогового значения 10, приложение автоматически открепляется, если оно было закреплено. Если же вызвать функцию `tryUnsnap` из незакрепленного приложения, этот вызов ни к чему не приведет.

Используйте пропорциональный дизайн

В основе пропорционального дизайна лежит идея о том, что любой выводимый на экран контент должен визуализироваться с размерами, пропорциональными ширине элемента-контейнера. Пропорциональный макет, который также часто называют *текучим* (liquid), или *резиновым* (fluid), автоматически придает контенту приложения способность приспосабливаться к любому размеру.

В контексте пропорционального макета вы не станете, к примеру, давать элементу `div` фиксированный размер, как мы делали это в коде, результат выполнения которого показан на рис. 8.8. И тем не менее нет никаких гарантий того, что любое приложение, даже имеющее пропорциональный макет, можно будет качественно визуализировать на пространстве шириной 320 пискелов.

Разработчики приложений Магазина Windows стремятся задействовать такой пользовательский интерфейс, который функционирует на основе списочных и плавающих элементов. Нижележащая платформа содействует этому посредством таких специализированных компонентов, как `ListView` и `SemanticZoom`, с которыми мы работали в предыдущей главе. Любой контент, выводимый на экран с помощью этих компонентов, имеет хорошие шансы на то, чтобы достойно выглядеть на представлении в режиме закрепления.

Давайте изучим пропорциональные макеты более подробно.

Текучие макеты

Душой текучих макетов является способность HTML-элементов (таких, как изображения, контейнеры, текст) сохранять свои положение и размер относительно друг друга и относительно экрана. В дополнение к способности правильно менять размеры и масштабировать шрифты еще одной важной способностью текучих макетов является такое управление свободным местом, при котором элементы пропорционально распределяются по всему доступному пространству.

Windows 8 превосходно поддерживает подход, де факто ставший уже стандартом. Этот подход основан на концепции *гибких блоков* (flexible boxes). Дополнительные сведения можно найти по адресу http://bit.ly/SUM20b.

Гибкие блоки

В приложениях Магазина Windows гибкий блок создается путем назначения элементу `div` конкретного набора CSS-стилей. Чтобы попрактиковаться, откроем файл default.css проекта SnapMe и добавим следующий код:

```
.flexible-container {
   display: -ms-flexbox;
   -ms-flex-direction: row;
   -ms-flex-align: start;
   -ms-flex-wrap: wrap;
   color: white;
   font-size: 2em;
   text-align: center;
   height: 400px;
   overflow: auto;
   margin-top: 10px;
 }
 #block1 {
   background: #43e000;
   padding: 10px;
   border: solid 2px #fff;
 }
 #block2 {
   background: #166aff;
   padding: 20px;
   border: solid 2px #fff;
 }
 #block3 {
   background: #43e000;
   padding: 20px;
   border: solid 2px #fff;
 }
 #block4 {
   background: #abababab;
   padding: 25px;
   border: solid 2px #fff;
 }
 #block5 {
   background: #ff6a00;
   padding: 10px;
   border: solid 2px #fff;
}
```

Теперь вернемся к файлу default.html и добавим туда разметку, включающую в себя элементы, стиль которым назначается с помощью предыдущих классов таблицы стилей:

```
<div id="flexBox" class="flexible-container">
   <div id="block1">Europe</div>
   <div id="block2">North America</div>
   <div id="block3">Australia</div>
   <div id="block4">Asia and Far East</div>
   <div id="block5">South America</div>
</div>
```

Теперь у вас есть контейнер — элемент `div` с именем `flexBox`, способный сделать плавающим свой контент, состоящий из пяти дочерних элементов с именами `block`*N*

в пределах доступного пространства. Главное преимущество такого подхода заключается в том, что контент не обрезается. Кроме того, дочерние блоки будут переноситься на новое место и выравниваться в соответствии со значениями атрибутов `-ms-flex-`*XXX* CSS-класса `flexible-container`. Однако ключевым стилем является значение `-ms-flexbox`, присвоенное атрибуту `display` CSS-класса `flexible-container`: именно этот атрибут делает контент элемента `div` плавающим вертикально или горизонтально и согласующимся с другими параметрами.

В табл. 8.1 представлен краткий перечень вариантов настройки, имеющихся в вашем распоряжении для задания параметров визуализации дочерних элементов в гибком блоке.

Таблица 8.1. Стили, поддерживаемые в гибком блоке

Стиль	Описание
-ms-flex-direction	Ориентация дочерних элементов внутри гибкого блока. Возможные значения: row (строка), column (столбец), row-reverse (реверс строки) и column-reverse (реверс столбца). Значением, предлагаемым по умолчанию, является row. Это значение указывает на то, что дочерние элементы, чтобы заполнить пространство, выстраиваются по горизонтали. Элементы перечисляются в порядке их объявления в исходном коде. В отличие от этого, значение column заставляет элементы выстраиваться вертикально. Классификатор reverse используется для изменения порядка выстраивания элементов на обратный (от последнего к первому)
-ms-flex-align	Выравнивание дочерних элементов внутри гибкого блока. Выравнивание должно быть вертикальным, если их ориентация (стиль ms-flex-direction) горизонтальная, и горизонтальным, если их ориентация вертикальная. Допустимые значения: start, end, center и stretch (используется по умолчанию). Значение start приводит к выравниванию элементов по верхнему (или по левому) краю. Значение end приводит к выравниванию элементов по нижнему (или по правому) краю. Значение center приводит к центрированию элементов, а значение stretch придает всем элементам максимально возможную ширину или высоту
-ms-flex-pack	Порядок распределения излишков пространства между дочерними элементами в гибком блоке. Допустимые значения: start, end, center и justify (используется по умолчанию). Значение start оставляет все пространство в конце строки или столбца. Значение end оставляет все пространство в начале строки или столбца. Значение center просто центрует элементы, а значение justify разбивает лишнее пространство между дочерними элементами
-ms-flex-wrap	Перенос дочерних элементов на несколько строк или столбцов на основе объема доступного пространства в объекте. Допустимые значения: none (используется по умолчанию), wrap и wrap-reverse. Значение none указывает на то, что каждый дочерний элемент помещается в отдельной строке или в отдельном столбце. А значение wrap заставляет гибкий блок размещать дочерние элементы последовательно, в порядке их объявления. В отличие от него, значение wrap-reverse делает порядок следования элементов обратным

Теперь попробуйте откомпилировать и запустить приложение. При работе приложения в режиме заполнения вы должны получить результат, показанный на рис. 8.9.

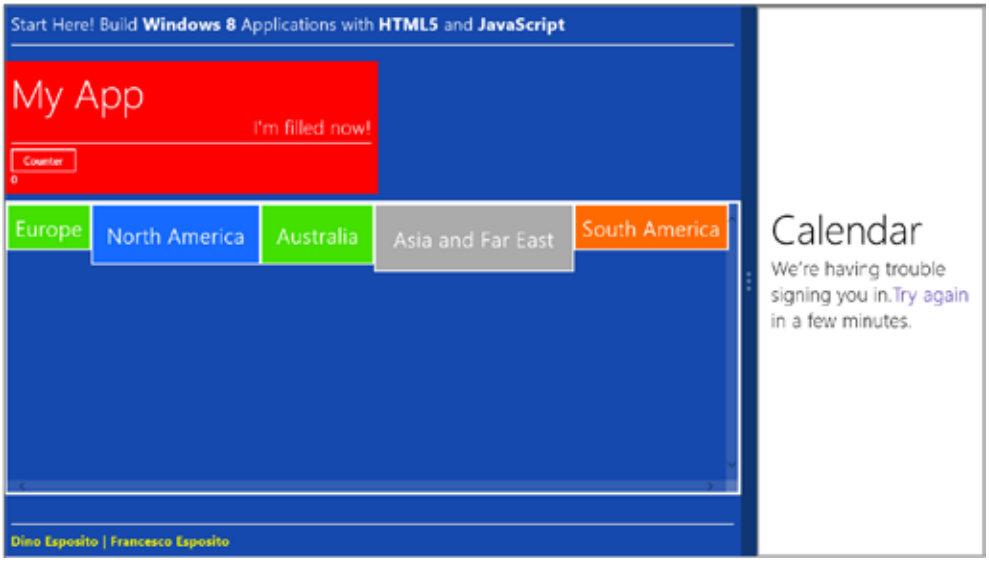

Рис. 8.9. Приложение SnapMe в режиме заполнения

Как видите, все блоки внутри гибкого контейнера выведены горизонтально и хорошо вписаны в доступное пространство. Если приложение будет визуализировано в полноэкранном режиме с альбомной ориентацией, контейнер растянется на всю ширину, а лишнее пространство окажется в конце строки. Такой порядок обусловливается значением `start`, присвоенным атрибуту `-ms-flex-align`.

В отличие от этого, на рис. 8.10 показано приложение, закрепленное у левого края экрана. Доступного пространства теперь слишком мало, чтобы вместить в себя несколько блоков в одной строке. В результате элементы размещаются вертикально. Если элемент окажется слишком большим и не поместится на пространстве шириной 320 пикселов, у вас будут два варианта действий: смириться с тем, что лишний контент окажется обрезанным, или сделать гибкий контейнер прокручиваемым. Это можно сделать, добавив к гибкому контейнеру следующий стиль:

```
.flexible-container {
   overflow: scroll;
   ...
}
```

Обратите внимание на то, что в упражнении мы уже включили эти инструкции в код, добавленный ранее в файл default.css.

И наконец, на рис. 8.11 можно познакомиться с гибким контейнером в полноэкранном режиме с книжной ориентацией.

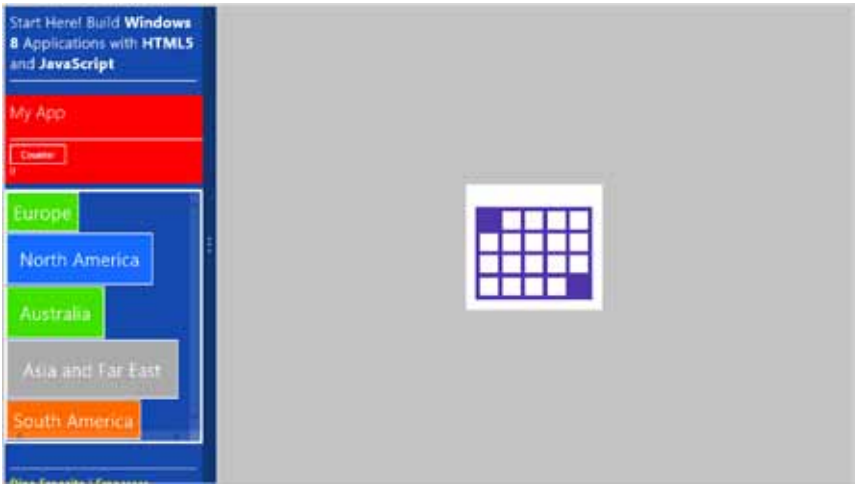

Рис. 8.10. Приложение SnapMe в режиме закрепления

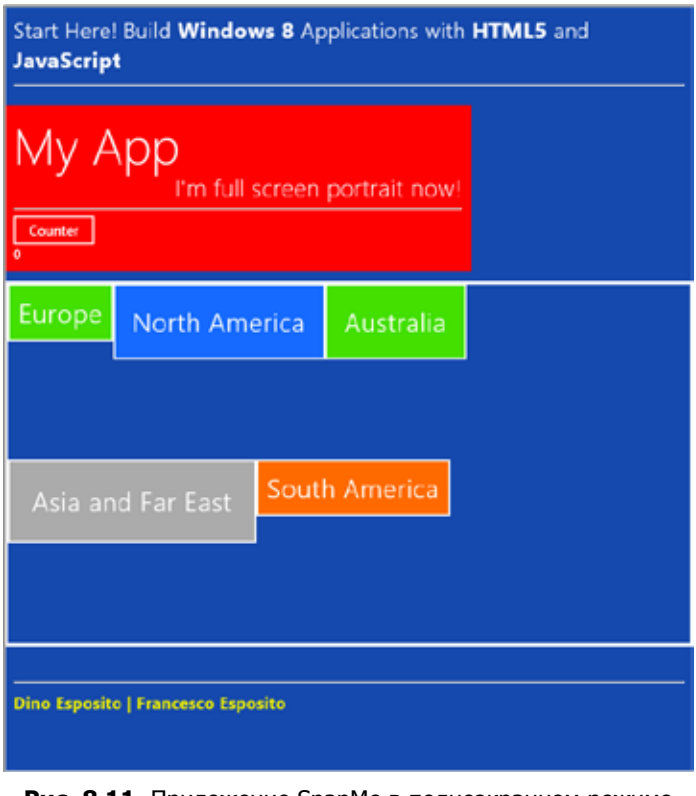

Рис. 8.11. Приложение SnapMe в полноэкранном режиме с книжной ориентацией

Медиазапросы CSS-файлов

Гибкий контейнер является весьма мощным средством, положенным в основу технологии CSS, на которой базируются компоненты, специфичные для Windows 8, такие как `GridView`, `ListView` и `SemanticZoom`. Теперь представьте, что вы увеличиваете размер шрифта для всех дочерних элементов. Это легко сделать, если открыть файл default.css и внести в имеющий в нем класс `flexible-container` следующие изменения:

```
.flexible-container {
   font-size: 3em;     // Прежнее значение было 2em
   ...
}
```

Рис. 8.12. Более крупный шрифт приводит к появлению в режиме закрепления полос прокрутки

Проблем может не возникнуть, если приложение не перейдет в режим закрепления (рис. 8.12).

Полосы прокрутки необходимы для того, чтобы увидеть не помещающийся контент. А есть ли способ автоматической настройки некоторых CSS-стилей, когда приложение переходит в режим закрепления? Конечно, есть, именно для этого и предназначены медиазапросы CSS-файлов, о которых уже вкратце упоминалось в главе 3.

Медиазапрос выдается программой при определенных условиях и заставляет браузер выбирать конкретный CSS-файл. Медиазапрос можно определить с помощью элемента `link` (о чем уже упоминалось в главе 3) или путем его непосредственной вставки в CSS-файл. Это стандартный подход, которому следуют приложения Магазина Windows.

При создании нового проекта в файл default.css всегда включается следующий код:

```
@media screen and (-ms-view-state: fullscreen-landscape) {
 }
 @media screen and (-ms-view-state: filled) {
 }
 @media screen and (-ms-view-state: snapped) {
 }
 @media screen and (-ms-view-state: fullscreen-portrait) {
}
```

Это просто предопределенные медиазапросы для четырех визуальных состояний приложения Магазина Windows.

Вам остается только добавить CSS-параметры, которые нужно применить к каждому конкретному визуальному состоянию. Например, чтобы размер шрифта уменьшался,

только когда приложение переходит в режим закрепления, нужно ввести в файл default.css следующий код:

```
@media screen and (-ms-view-state: snapped) {
    .flexible-container {
      font-size: 2em;
    }
}
```

По сути, вы назначаете любым существующим стилям значения, применяемые к конкретному сценарию. Полностью стиль переписывать не нужно, достаточно лишь указать различия.

Медиазапросы CSS-файлов или события изменения размера

В данной главе, для того чтобы справиться с изменениями режима просмотра, мы применяли два подхода: обнаружение событий изменения размера и медиазапросы CSS-файлов. Чем они отличаются друг от друга и когда применять каждый из них?

Медиазапросами CSS-файлов управляет браузер, в задачу которого входит применение к приложению тех или иных таблиц CSS-стилей. Если потребности приложения ограничиваются только изменением размеров шрифтов, настройкой некоторых фиксированных параметров ширины или высоты либо корректировкой полей и отступов, предпочтительнее и эффективнее использовать медиазапросы CSS-файлов. Кроме того, медиазапросы позволяют скрывать и перемещать элементы.

Однако могут возникать такие ситуации, когда нельзя просто скорректировать разметку приложения, ограничившись только возможностями CSS. В качестве типичного примера можно привести использование для организации списка компонента `ListView`. В режиме просмотра с достаточно большим пространством удобнее выбрать макет с несколькими столбцами. Однако в режиме закрепления целесообразно перейти к макету с одним столбцом. Поскольку `ListView` является компонентом Windows 8, для перехода к нужному макету требуется JavaScript-код. Из CSS-файла это сделать невозможно, а единственным доступным подходом является использование обработчика события. С помощью событий можно также программным способом полностью (или частично) заменить макет приложения, выбирая другой макет для каждого конкретного визуального состояния. Вскоре мы выполним соответствующее упражнение.

ВНИМАНИЕ

Отсюда следует вывод о том, что в первую очередь нужно остановить свой выбор на медиазапросах CSS-файлов, а к событиям переходить, только если возможностей медиазапросов окажется недостаточно. Тем не менее следует заметить, что в действительности довольно трудно найти более-менее серьезное приложение, визуальными состояниями которого можно управлять без обращения (хотя бы в некоторых случаях) к обработке событий.

Адаптация видеоприложений

В главе 7 мы создали видеоприложение, в котором компонент `ListView` давал пользователям возможность выбрать с YouTube видеоклип, который затем воспроизводился с помощью элемента `iframe`. Это приложение было написано при полном игнорировании визуальных состояний, поэтому, как показано на рис. 8.13, ждать от него нужной реакции на изменение визуального состояния не приходится.

Рис. 8.13. Видеоприложение, созданное в главе 7, при работе в режиме заполнения обрезается

При работе в режиме заполнения у приложения обрезается существенная часть пользовательского интерфейса — собственно видеоплеер. Нетрудно понять, что в режиме закрепления или с книжной ориентацией, когда ширина экрана оказывается еще меньше, все может быть только хуже. Чтобы видеоприложение хорошо вписывалось в окружающую среду, нужно внести в него ряд изменений.

Пользовательский интерфейс создан с применением двух основных элементов `listview`, предназначенных для выбора видео и плеера. Первое, что можно сделать — поместить оба элемента в гибкий контейнер. Затем можно избавиться от основной части фиксированных значений для размеров, заданных в главе 7. В принципе, явно указывать значения ширины и высоты нужно только в следующих случаях:

- при определении высоты элементов `listview`, когда пользователь выбирает видеоклип для воспроизведения;
- при определении ширины элементов `listview`, чтобы получить хороший внешний вид в режиме закрепления;
- при определении ожидаемого размера видеоплеера.

Поэтому откройте файл default.css, пропустите классы, определенные для тела, заголовка, нижнего колонтитула и названия, а затем удалите все остальное. Далее добавьте следующий код:

```
.flexible-container {
   display:-ms-flexbox;
   border: solid 1px #fff;
   -ms-flex-pack: center;
   -ms-flex-wrap: wrap;
 }

 /* Шаблон для заголовков в увеличенном компоненте ListView */
 .header-title {
   padding: 8px;
 }

 /* Шаблон для позиций списка в увеличенном компоненте ListView */
 .zoomed-in-item-container {
   overflow: hidden;
   display: -ms-grid;
 }
 .zoomed-in-item-container img {
   -ms-grid-column: 1;
 }
 .zoomed-in-item-container div {
   -ms-grid-column: 2;
 }

 /* Шаблон для позиций списка в уменьшенном компоненте ListView */
 .zoomed-out-item-container {
   background-color: #00f;
   padding: 8px;
   text-align: center;
   width: 320px;
 }
 .zoomed-out-item-container h1 {
   color: #fff;
   font-size: 3em;
 }

 /* CSS-стили для увеличенных (уменьшенных) компонентов ListView */
 #listview-in {
 }
 #listview-out {
 }

 /* Общий контейнер */
 #semantic-zoom-container {
   height: 240px;
   width: 100%;
   border-top: solid 2px #31cfd4;
   border-bottom: solid 2px #31cfd4;
   background: -ms-radial-gradient(
     center, ellipse cover, #c5deea 0%,#8abbd7 31%,#066dab 100%);
 }
```

продолжение ⇩

```
 /* Видеоплеер */
 #player-container {
   height: 338px;
   width: 450px;
   border: solid 1px #111;
   background: -ms-linear-gradient(
     left, #111 0%,#444 50%,#444 51%,#111 100%);
}
```

Как видите, для компонента `listview` задана высота 240 пикселов, а для позиций списка — ширина 320 пикселов, что хорошо подходит для визуализации в режиме закрепления только по одной позиции в строке. Теперь откройте файл default.html и внесите в разметку правку, удалив весь элемент `table`, расположенный сразу за элементом `h1` с названием страницы, и вставив вместо него следующую разметку:

```
<div class="flexible-container">
   <div id="semantic-zoom-container"
     data-win-control="WinJS.UI.SemanticZoom"
     data-win-options="{initiallyZoomedOut: true}">

     <!-- Увеличенное представление. -->
     <div id="listview-in"
 data-win-options ="{layout: {type: WinJS.UI.GridLayout}}"
 data-win-control="WinJS.UI.ListView"></div>

     <!-- Уменьшенное представление. -->
     <div id="listview-out"
       data-win-options ="{layout: {type: WinJS.UI.GridLayout}}"
       data-win-control="WinJS.UI.ListView"></div>
   </div>
   <div id="player-container">
     <iframe id="player" height="100%" width="100%" type="text/html"></iframe>
   </div>
</div>
```

И наконец, откройте файл Videos.js и добавьте к функции `VideosApp.init` такую строку:

```
// Регистрация обработчика события изменения размера
window.onresize = addEventListener('resize', VideosApp.onResize, false);
```

Эта дополнительная строка зарегистрирует обработчик события изменения размера. Обрабатывая это событие, можно переключать макет обоих элементов `listview` компонента семантического масштабирования на макет списка, предназначенный для работы приложения в режиме закрепления. Код для функции `VideosApp.onResize` имеет следующий вид:

```
VideosApp.onResize = function (e) {
   var detailView = document.getElementById("listview-in").winControl;
   var masterView = document.getElementById("listview-out").winControl;
   var viewState = Windows.UI.ViewManagement.ApplicationView.value;

   switch (viewState) {
```

```
      case Windows.UI.ViewManagement.ApplicationViewState.snapped:
        detailView.layout = new WinJS.UI.ListLayout();
        masterView.layout = new WinJS.UI.ListLayout();
        break;
      case Windows.UI.ViewManagement.ApplicationViewState.filled:
      case Windows.UI.ViewManagement.ApplicationViewState.fullScreenLandscape:
      case Windows.UI.ViewManagement.ApplicationViewState.fullScreenPortrait:
        detailView.layout = new WinJS.UI.GridLayout();
        masterView.layout = new WinJS.UI.GridLayout();
        break;
    }
}
```

Сетка предоставляет компонентам макет с несколькими колонками, в котором позиции списка выводятся на экран по вертикали, пока не достигнут дна, а затем переносятся к следующей колонке. В режиме закрепления в условиях ограниченности горизонтального пространства лучше выбрать такой макет списка, в котором создается одна колонка.

Новое приложение показано на рис. 8.14. Такое приложение остается полностью работоспособным в любом визуальном состоянии.

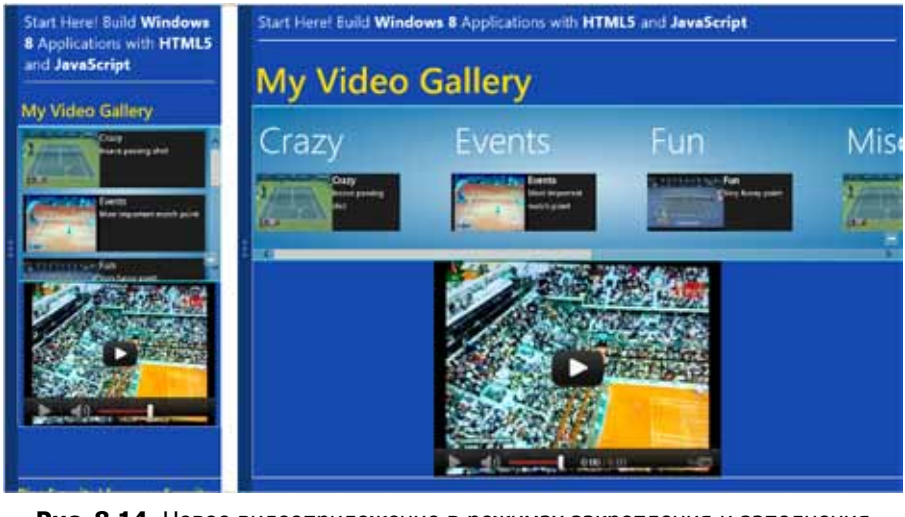

Рис. 8.14. Новое видеоприложение в режимах закрепления и заполнения

Ранее для ширины плеера было установлено значение 400 пикселов. На рис. 8.14 показано, что при работе приложения в режиме закрепления плеер все равно виден целиком. Этот эффект получен в результате еще одного изменения: добавления медиазапроса для состояния закрепления:

```
@media screen and (-ms-view-state: snapped) {
   #player-container {
     height: 225px;
```

продолжение ⇩

```
      width: 300px;
    }
}
```

Когда приложение закрепляется у края экрана, медиазапрос обеспечивает выбор для плеера меньшего размера.

ПРИМЕЧАНИЕ

В этой главе мы практиковались только в создании одностраничных приложений. Однако следует заметить, что учитывать визуальные состояния нужно для любой страницы, имеющейся в приложении.

Выводы

Приложения Магазина Windows могут визуализироваться в четырех различных режимах: в полноэкранном с альбомной ориентацией, в полноэкранном с книжной ориентацией, а также в режимах заполнения и закрепления. Из этого следует, что макет вашего приложения нужно продумывать как минимум для четырех различных сценариев с прицелом на сохранение его функциональности, даже если оно окажется в маленьком контейнере размером 320 × 760 пикселов. В то же время при создании приложения для Windows 8 нужно учитывать вероятность его просмотра на очень больших экранах.

Поэтому разработчикам очень важно с самого начала выбрать такой макет, который легко расширяется. Если же выбрать единый гибкий макет не представляется возможным, нужно спланировать несколько различных макетов и программным способом переключаться между ними при обнаружении события изменения размеров.

Большой интерес в этом плане представляют приложения, закрепленные у края экрана шириной всего 320 пикселов. В руководствах по правильному программированию приложений для Магазина Windows сказано, что нужно стремиться к достижению одинаковой функциональности приложения в разных визуальных состояниях. Но это не обязательное требование, а лишь вектор, указывающий нужное направление. Хотя пользователи вполне резонно полагают, что любое приложение можно закрепить у края экрана, не все приложения способны полностью сохранить свою функциональность на пространстве шириной всего 320 пикселов. На этот случай вы должны быть готовы предложить альтернативный макет и программным способом вывести приложение из режима закрепления при выборе пользователем такой функции, для работы которой требуется весь экран.

В данной главе мы узнали о функционировании приложений в режимах закрепления и заполнения, а также поэкспериментировали с кодом и стилями, упрощающими создание таких приложений Магазина Windows, которые смогут стать достойными представителями мира Windows 8.

9 Интеграция в среду Windows 8

У сердца свои законы, разуму неподвластные.

Блез Паскаль

Операционные системы предназначены главным образом для того, чтобы обеспечивать успешное выполнение приложений. Существует множество критериев оценки успешности приложения. Однако, с точки зрения пользователей, успешное приложение — это чаще всего просто приложение, позволяющее пользователю решать обычные задачи простым и удобным способом.

Рассмотрим приложение, которое должно сохранить какой-то текст в файле на диске. Основным предназначением приложения является получение и последующее сохранение текста. Однако для сохранения текста пользователю, вероятно, нужно выбрать имя файла и его место на диске. Для приложения это второстепенные задачи, не относящиеся к его основному предназначению. Тем не менее вам как разработчику в любом случае придется тратить свое время и создавать код для их решения.

Так какой же код нужен для решения таких вспомогательных задач, как выбор файла на диске? Если бы не было вполне определенных правил, каждое приложение могло бы предлагать для этого собственный пользовательский интерфейс. Это совершенно неприемлемо для пользователей, которые сталкивались бы с потенциально несметным количеством разных интерфейсов, но это также неприемлемо и для разработчиков, которым пришлось бы всякий раз писать новый код.

В данной главе вы узнаете о *контрактах* (contracts) и приемах работы с ними. Контракты — это специальные сервисы для решения обычных задач, которые

приложения могут использовать и предоставлять для использования другим приложениям. Конечной целью контрактов является достижение такого положения вещей, при котором все больше и больше приложений производят одинаковые действия одинаковым образом.

Контракты и обычные задачи

Операционная система Microsoft Windows 8 накладывает на приложения ряд ограничений. В частности, приложения изолируются друг от друга. Но как же тогда приложения могут взаимодействовать друг с другом? Ответ — посредством контрактов.

Контракт определяет общесистемный протокол взаимодействия и совместного использования данных для приложений, разрабатываемых разными компаниями. Контракт касается одной конкретной задачи, например: совместного использования данных, поиска данных, выбора файлов, установки параметров и т. д. Приложения, нуждающиеся в этом конкретном поведении, могут без создания собственного кода опираться на уже существующие реализации контрактов. В то же время, приложения могут сами предоставить тот или иной контракт, чтобы им могли воспользоваться другие приложения.

Контракты в Windows 8

Существует три аспекта применения контрактов, заслуживающие более пристального внимания: как находить доступные сервисы, как использовать сервисы и как предоставлять сервисы.

Выдвигающаяся панель

В Windows 8 выдвигающаяся панель (charms bar) представляет собой системную панелью инструментов, доступ к которой пользователь может получить в любое время независимо от того, какое из приложений активно в данный момент. Эта панель выдвигается с правой стороны экрана, когда указатель мыши наводится на верхний правый или нижний правый угол экрана. Выдвигающуюся панель можно также вывести на экран, нажав сочетание клавиш Windows+C. И наконец, на сенсорных устройствах эта панель вызывается проведением пальцем по экрану с правой стороны. Выдвигающаяся панель показана на рис. 9.1.

Выдвигающаяся панель состоит из пяти элементов: Search (Поиск), Share (Обмен данными), Settings (Настройки), Start (Пуск) и Devices (Устройства). С точки зрения контрактов наиболее интересными являются первые три элемента. По сути, элементы Search (Поиск), Share (Обмен данными) и Settings (Настройки) являются тремя базовыми контрактами, которые должны, как минимум, приниматься во внимание большинством приложений Магазина Windows.

Рис. 9.1. Выдвигающаяся панель в действии

Приложение, которому требуется поделиться частью своего контента с другими приложениями (и, стало быть, с другими пользователями), реализует контракт обмена данными. Пользователи операционной системы Windows 8 знают, что у них есть стандартное средство доступа к любому контенту, который может иметься у текущего приложения — это кнопка Share (Обмен данными) на выдвигающейся панели. Путем реализации контракта обмена данными приложение может предоставить свой выбранный контент любому приложению, упомянутому в списке кнопки Share (Обмен данными) на выдвигающейся панели.

Точно так же приложение, дающее пользователям возможность вести поиск какого-нибудь контента (например, поиск конкретного рецепта приготовления блюда в приложении для кулинаров), не нуждается в специальном пользовательском интерфейсе поиска. Пользователи знают, что для них существует стандартное средство поиска контента на выдвигающейся панели. Поэтому приложение, поддерживающее контракт поиска, может предлагать для поиска свой контент всегда, когда пользователь обращается к элементу Search (Поиск) на выдвигающейся панели.

И наконец, кнопка Settings (Настройки) открывает доступ к странице настройки текущего приложения. Путем реализации контракта настройки приложение может предоставить собственное меню и набор элементов управления на панели Settings (Настройки) во многом так же, как это делает, например, Microsoft Internet Explorer (рис. 9.2).

Контракты поиска, обмена данными и настройки — это лишь основные контракты, доступные в Windows 8. Далее в этой главе вы узнаете, что есть еще и другие доступные контракты.

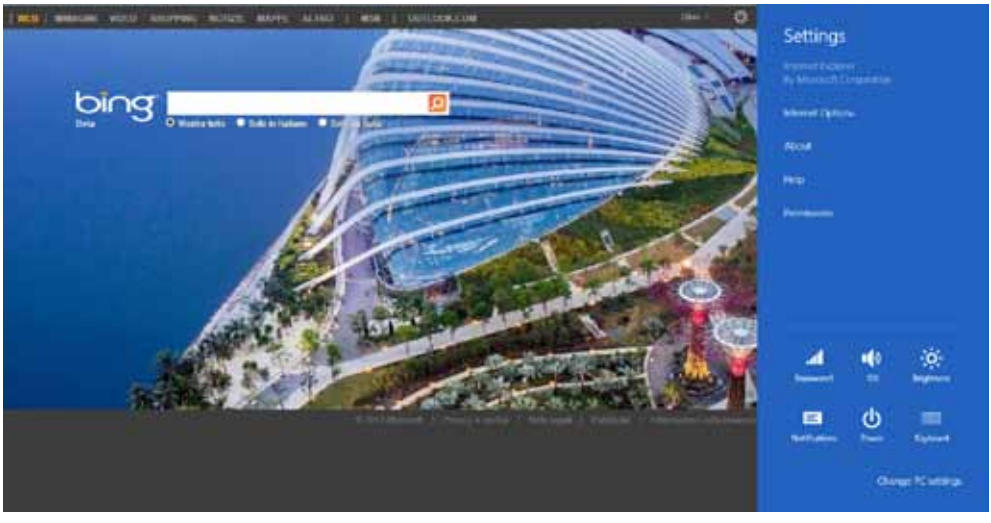

Рис. 9.2. Панель настройки в Internet Explorer

Потребление и предоставление сервисов

Приложения Магазина Windows могут работать с контрактами двумя способами: использовать сервис и предоставлять сервис. С точки зрения программирования реализовать первый сценарий намного проще, так как для этого достаточно задействовать уже готовый набор компонентов Windows 8. Второй сценарий программировать несколько сложнее, поскольку для этого нужно, чтобы приложение соответствовало правилам, установленным в определении контракта.

Наше приложение TodoList со списком заданий, чтобы дать возможность совместно использовать некоторые свои задания с друзьями, может реализовать в качестве источника данных контракт обмена данными. Пользователь сможет вызвать выдвигающуюся панель и выбрать на панели Share (Обмен данными), скажем, приложение Mail (Почта). Затем это приложение сможет получить контент, предоставляемый приложением TodoList, и подготовить с его помощью почтовое сообщение.

В то же время ваше приложение может работать в качестве приемника той информации, которой решит поделиться другое приложение. Предположим, у вас есть приложение, выводящее сообщение на стену Фейсбука пользовательской учетной записи. Создавая в приложении в качестве потребителя реализацию контракта обмена данными, вы даете пользователям других приложений возможность отправлять вашему приложению тот контент, которым они хотят поделиться. К примеру, пользователи приложения TodoList смогут теперь публиковать свои задания непосредственно в своих учетных записях Фейсбука.

Особую привлекательность контрактам придает тот факт, что приложения Магазина Windows в конечном итоге могут объединять в себе функциональные возможности

разных приложений. Заимствуя нужную функциональность у других приложений, разрабатывать приложения проще и быстрее.

Контракты и расширения

В этой главе мы выполним несколько упражнений для изучения наиболее популярных контрактов. Назначение данной главы состоит в том, чтобы вы, кратко познакомившись с перечнем доступных контрактов, знали, где и что искать, когда вам потребуется реализовать или использовать тот или иной сервис.

Поддерживаемые контракты

В табл. 9.1 перечислены контракты, которые могут поддерживаться приложениями Магазина Windows. Как видите, некоторые контракты, которые мы до сих пор знали под одним именем (например, контракт обмена данными), могут быть доступны для программирования в двух и более разновидностях (например, контракты источника и приемника общих данных).

Таблица 9.1. Контракты, доступные приложениям Магазина Windows

Контракт	Описание
Share source (Источник общих данных)	Приложение реализует этот контракт для совместного использования части своего контента с любыми зарегистрированными приложениями, поддерживающими контракт Share target (Приемник общих данных). Приложения-приемники указываются в меню Share выдвигающейся панели. Например, доступное в Windows 8 по умолчанию приложение Weather (Погода) может делиться прогнозами погоды
Share target (Приемник общих данных)	Приложение реализует этот контракт, чтобы попасть в список приложений выдвигающейся панели, способных получать любые данные, которыми может поделиться текущее приложение. Например, имеющееся в системе приложение Mail (Почта) работает в качестве потребителя общих данных
File open/Save picker (компонент выбора открываемого или сохраняемого файла)	Приложение реализует этот контракт, чтобы предоставить собственное средство просмотра файлов и соответствующих папок. Этот контракт предусматривает предоставление пользовательского интерфейса для выбора файла или папки
Settings (Настройки)	Приложение реализует этот контракт, чтобы предоставить пользователям доступ к настройке приложения, определяющей способы взаимодействия пользователей с этим приложением. Windows 8 требует, чтобы приложения предоставляли средства настройки стандартным образом — через реализацию контракта Settings (Настройки)

продолжение ⇩

Таблица 9.1 (*продолжение*)

Контракт	Описание
Search (Поиск)	Приложение реализует этот контракт, чтобы дать возможность пользователям вести поиск внутри любого доступного контента, характерного для приложения
Play to (Воспроизведение)	Приложение реализует этот контракт, чтобы дать возможность пользователям легко и просто организовывать поток аудио- и видеоданных или изображений из их компьютера на любое подключенное устройство своей домашней сети (например, на большой экран телевизора)
Cache updater (Средство обновления кэша)	Приложение реализует этот контракт, если предоставляет регулируемый доступ к файлам и папкам (через контракт File picker provider) и нуждается в оповещении приложений об изменениях в перечисленных файлах и папках

Как видите, не все поддерживаемые контракты имеет смысл реализовывать в каждом приложении Магазина Windows. Например, контракты поиска и настройки требуются намного чаще, чем контракт воспроизведения или обновления кэша. Между тем среда Windows 8 предоставляет широкий выбор средств настройки, не ограничиваясь одними только контрактами.

Расширения

Если контракты касаются соглашений между приложениями, то расширения относятся к соглашениям между приложением и операционной системой Windows. Регистрируя расширение, приложение собирается дополнить или настроить одну или несколько стандартных функциональных возможностей Windows, например способ обработки в операционной системе файлов заданного формата (скажем, TXT-файлов). Расширения, доступные в Windows 8, перечислены в табл. 9.2.

Таблица 9.2. Расширения, доступные в приложениях Магазина Windows

Расширение	Описание
Account picture provider (Поставщик рисунка учетной записи)	Приложение предоставляет это расширение, чтобы попасть в перечень панели управления Account Picture Settings (Настройка рисунка учетной записи) в качестве приложения, которое может предоставить рисунок для учетной записи пользователя
AutoPlay provider (Поставщик автозапуска)	Приложение предоставляет это расширение, чтобы попасть в перечень приложений, выбранных для автозапуска при наступлении одного или нескольких событий AutoPlay. В Windows событие AutoPlay возникает, как только пользователь подключает устройство к компьютеру

Расширение	Описание
Background task provider (Поставщик фоновой задачи)	Приложение предоставляет это расширение, если ему нужно выполнить некоторую работу в фоновом режиме, когда его работа приостанавливается. Фоновые задачи должны быть небольшими и не требовать взаимодействия с пользователем
Camera settings provider (Поставщик настроек камеры)	Приложение предоставляет это расширение, если оно способно предложить нестандартный пользовательский интерфейс для настройки камеры и выбора эффектов, когда камера используется для фото- или видеосъемок
Contact provider (Поставщик контакта)	Приложение предоставляет это расширение, чтобы попасть в перечень приложений, которые Windows выводит на экран, когда пользователю нужно выбрать контакт
File activation provider (Поставщик активации файла)	Приложение предоставляет это расширение, если оно собирается зарегистрироваться в качестве обработчика файлов заданного формата
Print settings provider (Поставщик настроек печати)	Приложение предоставляет это расширение, если оно может предоставить нестандартный пользовательский интерфейс для задания параметров печати

Реализация контрактов и расширений требует следования четким правилам при написании кода и внесения некоторых изменений в файл манифеста приложения. Целью второго требования является информирование Windows об общесистемных изменениях, которые могут быть введены приложением.

Использование контракта выбора файла

В сохранении пользовательских данных на диске нуждается практически каждое приложение. При этом почти каждому приложению нужно создавать и открывать файлы в папках. Иногда приложение может использовать файлы и папки с фиксированными именами и местами на диске, а иногда предпочтительнее, чтобы последнее слово в отношении именования заданного файла и места размещения содержащей его папки оставалось за пользователем.

Но как дать пользователю возможность выбрать имя файла и его место на диске?

Именно для этого и предназначен контракт выбора файла. Если вы знакомы с более ранними версиями Windows, то для вас в концепции, заложенной в контракт выбора файла, не будет ничего нового — это всего лишь программный артефакт, необходимый для реализации обычного диалогового окна, похожего на то, что показано на рис. 9.3.

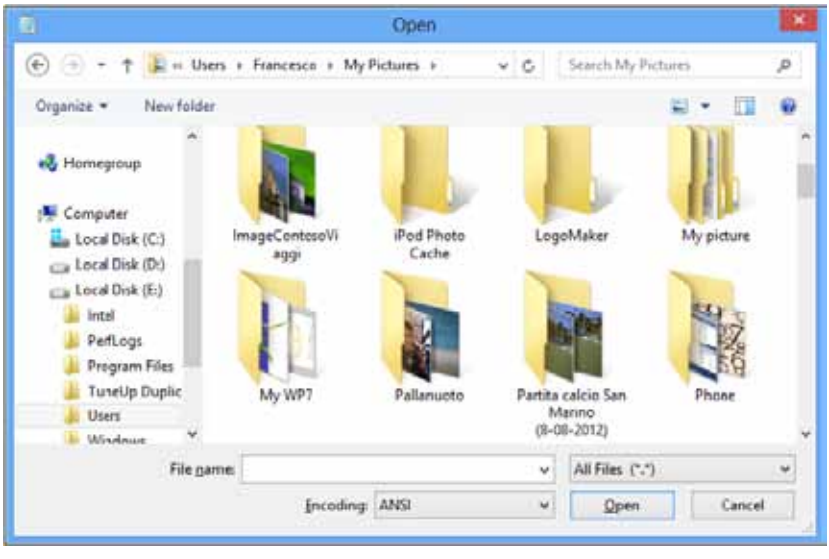

Рис. 9.3. Аналог средства выбора файла на рабочем столе Windows 7 и Windows 8

Давайте дополним наше приложение TodoList, которое мы разрабатывали в главе 6, чтобы дать пользователям возможность выбрать файл и место на диске для сохранения задания. Вам не придется разбираться с деталями, касающимися создания файла на физическом уровне. В этой главе мы ограничимся выбором файла, а завершить упражнение созданием файла и папки нам предстоит в следующей главе.

Выбор файла для сохранения данных

К выполнению упражнения нужно приступить с создания копии проекта TodoList из главы 6 в новой папке на ваш выбор. Перед тем как продолжить разбираться с контрактами, целесообразно внести ряд небольших изменений в существующий код, чтобы усовершенствовать приложение.

Предварительные изменения в приложении TodoList

В первую очередь нужно написать HTML-разметку для вывода сводной информации, показываемой пользователю перед сохранением данных. Откройте файл default.html и замените содержимое элемента `div` с идентификатором `flyoutSummary`.

Новый контент должен выглядеть следующим образом:

```
<div data-win-control="WinJS.UI.Flyout" id="flyoutSummary">
   <div class="tableLabel">DESCRIPTION:</div>
   <div class="tableValue"><span data-win-bind="innerText: description" /></div>
   <div style="clear:both" />
```

```
   <div class="tableLabel">DUE DATE:</div>
   <div class="tableValue">
     <span data-win-bind="innerText: dueDate TodoList.dateForDisplay"></div>
   <div style="clear:both" />
   <div class="tableLabel">PRIORITY:</div>
   <div class="tableValue"><span data-win-bind="innerText: priority" /></div>
   <div style="clear:both" />
   <div class="tableLabel">STATUS:</div>
   <div class="tableValue"><span data-win-bind="innerText: status" /></div>
   <div style="clear:both" />
   <div class="tableLabel">COMPLETED:</div>
   <div class="tableValue"><span data-win-bind="innerText: percCompleted" /></div>
   <div style="clear:both" />
   <br /><hr />
   <button onclick="TodoList.pickFileAndSaveTask()">It's OK. Please save!</button>
</div>
```

Как видите, теперь разметка содержит кнопку для пользователя, на которой он может щелкнуть, чтобы инициировать сохранение данных. Разметка содержит также несколько новых CSS-стилей, которые должны быть добавлены к имеющимся в проекте файлам todolist.css:

```
.tableLabel {
   float: left;
   width: 100px;
   text-align: right;
   font-weight: 600;
 }
 .tableValue {
   float: left;
   font-weight: 400;
   margin-left: 10px;
}
```

И наконец, нужно открыть файл todolist.js и внести изменения в функцию `displaySummary`:

```
TodoList.displaySummary = function (task) {
   // Подготовка контента для сводки
   var bindableElement = document.getElementById("flyoutSummary");
   WinJS.Binding.processAll(bindableElement, task);

   // Вывод сводки
   var anchor = document.getElementById("buttonAddTask");
   var flyoutSummary = document.getElementById("flyoutSummary").winControl;
   flyoutSummary.show(anchor);
}
```

Основные сведения, касающиеся привязки данных в Windows 8, вы получили в главах 5 и 6. В этой главе встретилось новое требование — необходимость форматирования данных для их вывода на экран. Задание, созданное с помощью приложения TodoList, имеет дату выполнения, а в сводке может потребоваться показать дату наряду со всем остальным контентом. При этом дату нужно вывести в общепринятом для даты формате. А получаемый исходный формат является форматом ISO

для дат, который человеку прочитать нелегко. Вам нужен конвертер, выполняющий преобразование даты непосредственно перед ее выводом на экран. В разметке для ранее введенного компонента `flyout` находится следующий код:

```
<span data-win-bind="innerText: dueDate TodoList.dateForDisplay">
```

Элемент `span` привязан к свойству `dueDate` объекта `Taskobject`, но манипулированием реальным контентом занимается функция `TodoList.dateForDisplay`, которую нужно добавить к файлу todolist.js:

```
// Конвертер даты, показываемой в сводке
 TodoList.dateForDisplay = WinJS.Binding.converter(function (value) {
   return value.toLocaleDateString();
});
```

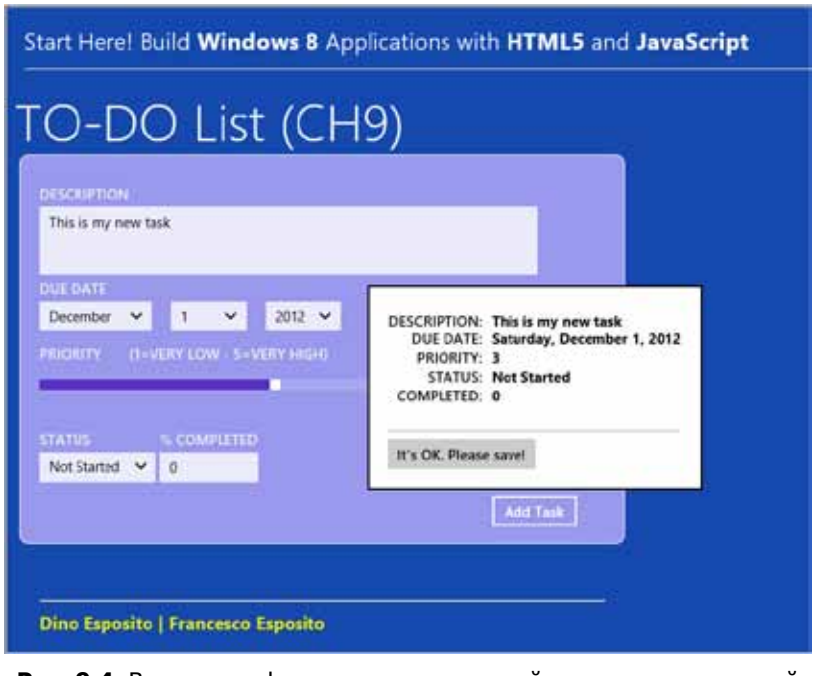

Рис. 9.4. Вывод модифицированного в данной главе окна со сводной информацией

И наконец, нужно добавить заглушку вместо кода, реализующего функциональность выбора файла. Этот код будет вызываться по щелчку пользователя на кнопке, которая теперь визуализируется в сводке и служит для того, чтобы инициировать сохранение. Добавьте в нижнюю часть файла todolist.js следующий код:

```
TodoList.pickFileAndSaveTask = function () {
   // Получение объекта задания для сохранения
   var currentTask = TodoList.getTaskFromUI();
```

```
    // Заглушка вместо более интересного кода
    TodoList.alert("Ready to pick a file and save...");
}
```

Новая сводная информация, показываемая пользователю, когда задание готово к сохранению, представлена на рис. 9.4.

Открепление приложения перед выбором файла

После того как пользователь щелкнет на кнопке, показанной на рис. 9.4, приложение должно позволить ему выбрать файл, в котором будет сохранено задание. Это влечет за собой использование контракта выбора файла в том виде, в котором он реализован операционной системой. Windows 8 предлагает собственную версию диалогового окна, показанного на рис. 9.3. Имеющийся в Магазине Windows аналог знакомого уже диалогового окна открытия-сохранения файла является просто компонентом, предоставляемым контрактом выбора файла.

На данном этапе вам пока не нужно подробно разбираться в контрактах, поскольку подробности контракта выбора файла скрыты в глубинах нескольких новых высокоуровневых компонентов и функций. Пока вам достаточно познакомиться только с этими компонентами и функциями.

Основное правило, требующее усвоения, заключается в том, что приложению Магазина Windows не разрешается вызывать компонент выбора файла, если оно находится в режиме закрепления. Поэтому в самое начало функции `TodoList.pickFileAndSaveTask` в файле todolist.js нужно добавить следующий код:

```
var currentState = Windows.UI.ViewManagement.ApplicationView.value;
 if (currentState === Windows.UI.ViewManagement.ApplicationViewState.snapped
    && !Windows.UI.ViewManagement.ApplicationView.tryUnsnap()) {
    // Если открепить приложение невозможно, выход по ошибке без уведомления
    return;
}
```

Если приложение не закреплено, можно вызывать системный компонент выбора сохраняемого файла.

Предлагаемый по умолчанию компонент выбора сохраняемого файла

Для выбора файла с целью его сохранения или чтения какого-нибудь контента Windows 8 предлагает два разных компонента. Если целью является выбор существующего файла для чтения его контента, предлагается объект `Windows.Storage.Pickers.FileOpenPicker`. А если вы собираетесь создать новый файл или перезаписать существующий, предлагается объект `Windows.Storage.Pickers.FileSavePicker`. Чтобы сохранить задание, нужно выбрать второй вариант. В функции `TodoList.pickFileAndSaveTask` из файла todolist.js должен быть следующий код:

```
TodoList.pickFileAndSaveTask = function () {
    // Код, проверяющий, закреплено ли приложение
    ...

    // Получение объекта задания для сохранения
    var currentTask = TodoList.getTaskFromUI();

    // Создание объекта выбора файла и установка параметров
    var savePicker = new Windows.Storage.Pickers.FileSavePicker();
    savePicker.commitButtonText = "Create task";
    savePicker.suggestedStartLocation =
      Windows.Storage.Pickers.PickerLocationId.computerFolder;
    savePicker.suggestedFileName = currentTask.description;
    savePicker.fileTypeChoices.insert("TodoList Task", [".todo"]);

    // А сюда помещается весь остальной код
    ...
}
```

Но этого кода для визуализации пользовательского интерфейса выбора файла еще недостаточно. В нем нет специальной инструкции, которая выводит на экран стандартный пользовательский интерфейс. Давайте потратим немного времени, чтобы разобраться в роли тех свойств, которые включены в представленный код.

Предпочтительные варианты настройки

Обратите внимание на то, что свойству `SuggestedStartLocation` присвоено выражение, указывающее на папку Computer как на место начала поиска. Как следует из названия свойства (`SuggestedStartLocation` — предлагаемое место начала поиска), это место не обязательно станет начальным местом при выборе файла. Пытаясь сохранить для пользователя привычную обстановку, компонент выбора файла узнает о последней просмотренной им папке и начнет свою работу именно с нее. Если же папка оказывается недоступной (например, после ее удаления) или недостижимой (например, после отключения от сети), компонент выбора файла принимает данное ему предложение. Предложение также принимается при отсутствии текущей записи с информацией о последней посещенной папке.

Следует также заметить, что разработчики не могут программным способом вносить предложения о начальной папке, если она не упомянута в перечне `PickerLocationId`. Конструктивно пользователю могут быть представлены только предопределенные системные папки. В перечень `PickerLocationId` входят папки, представленные в табл. 9.3.

Таблица 9.3. Предопределенные системные папки

Размещение	Описание
computerFolder	Папка, предоставляющая доступ ко всем дискам и подключенным устройствам
desktop	Папка рабочего стола Windows

Размещение	Описание
documentsLibrary	Папка документов пользователя
downloads	Предлагаемая по умолчанию папка для загрузки программ
homeGroup	Папка, предоставляющая доступ ко всем компьютерам домашней группы
musicLibrary	Папка музыкальной библиотеки
picturesLibrary	Папка библиотеки изображений
videosLibrary	Папка библиотеки видеоклипов

Можно также предложить имя сохраняемого файла. Для этого нужно воспользоваться свойством `SuggestedFileName`. В рассмотренном примере предлагаемое имя файла выбрано в соответствии с описанием созданного задания:

```
savePicker.suggestedFileName = currentTask.description;
```

Еще одним настраиваемым параметром является список расширений, рекомендуемых для создаваемого файла. В данном случае мы предлагаем собственное расширение .todo:

```
savePicker.fileTypeChoices.insert("TodoList Task", [".todo"]);
```

В свойстве `commitButtonText` задается надпись на кнопке, на которой пользователь должен щелкнуть для сохранения контента.

Получение имени создаваемого файла

Чтобы вывести на экран пользовательский интерфейс, с помощью которого пользователь мог бы выбрать файл для сохранения данных, вам нужно добавить к телу функции `TodoList.pickFileAndSaveTask` следующий код:

```
// Вызов компонента выбора файла
 savePicker.pickSaveFileAsync().then(function (file) {
   if (file) {
     TodoList.alert(file.name);
   }
});
```

Метод `pickSaveFileAsync` выводит на экран пользовательский интерфейс и возвращает управление только в том случае, если пользователь отклоняет диалог или выбирает файл. Этот метод выполняется в асинхронном режиме, а это означает, что для задания какого-либо поведения после выбора файла вам нужно воспользоваться методом `then`.

Компонент выбора сохраняемого файла возвращает параметр, указывающий на имя создаваемого файла. Если аргумент `file` имеет значение `null`, значит, пользователь отклонил диалог выбора файла, в противном случае пользователь успешно выбрал файл. Следующий шаг при выполнении этого упражнения заключается в выводе на экран некоторых свойств файла. Создавать и читать файлы вы научитесь в следующей главе. На рис. 9.5 показан пользовательский интерфейс выбора сохраняемого файла, готовый к созданию нового файла в пустой папке.

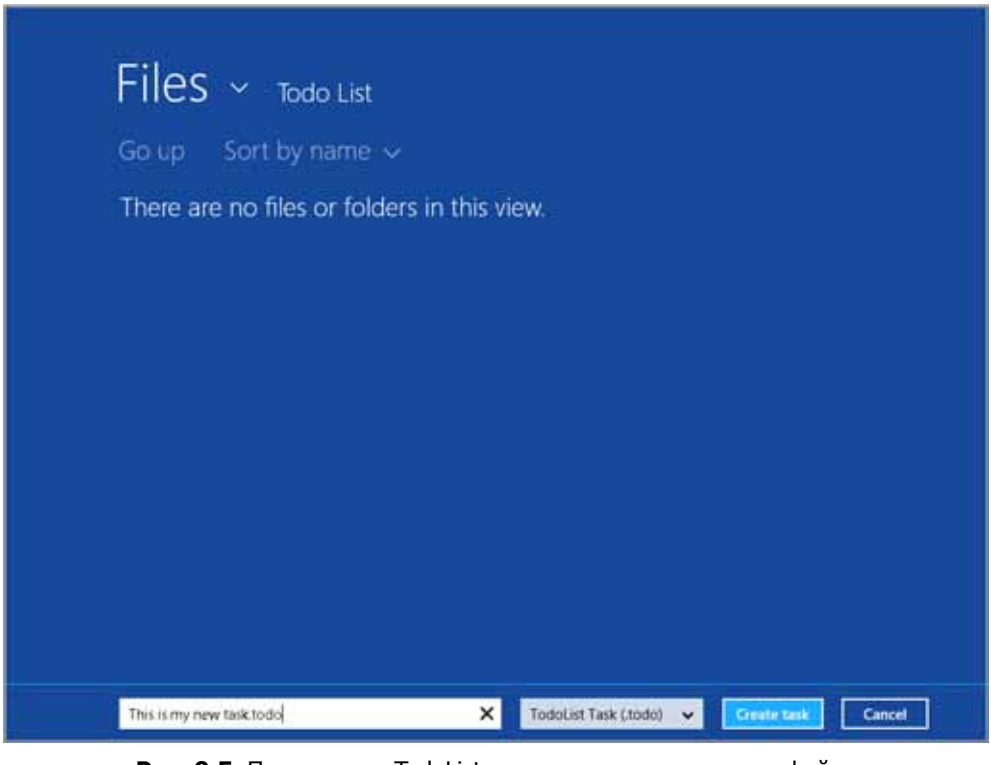

Рис. 9.5. Приложение TodoList готово к созданию нового файла в нестандартной папке TodoList

От компонента выбора файла вы получаете объект `StorageFile`. Для получения хранящегося в нем имени нужно вызвать свойство `name`. Чтобы избавиться от связанного с именем расширения, можно также вызвать свойство `displayName`. На рис. 9.6 показано окно сообщения с именем создаваемого файла.

Для получения имени или типа файла особых усилий не нужно. Вам доступны три свойства: `name`, `displayName` и `fileType`. Чтобы прочесть некоторые дополнительные свойства файла (например, размер или дату создания), придется вызвать функцию `getBasicPropertiesAsync`:

```
savePicker.pickSaveFileAsync().then(function (file) {
    if (file) {
      file.getBasicPropertiesAsync().then(function (basicProperties) {
        TodoList.alert(file.name + "(" + basicProperties.size + " bytes)");
      });
    }
});
```

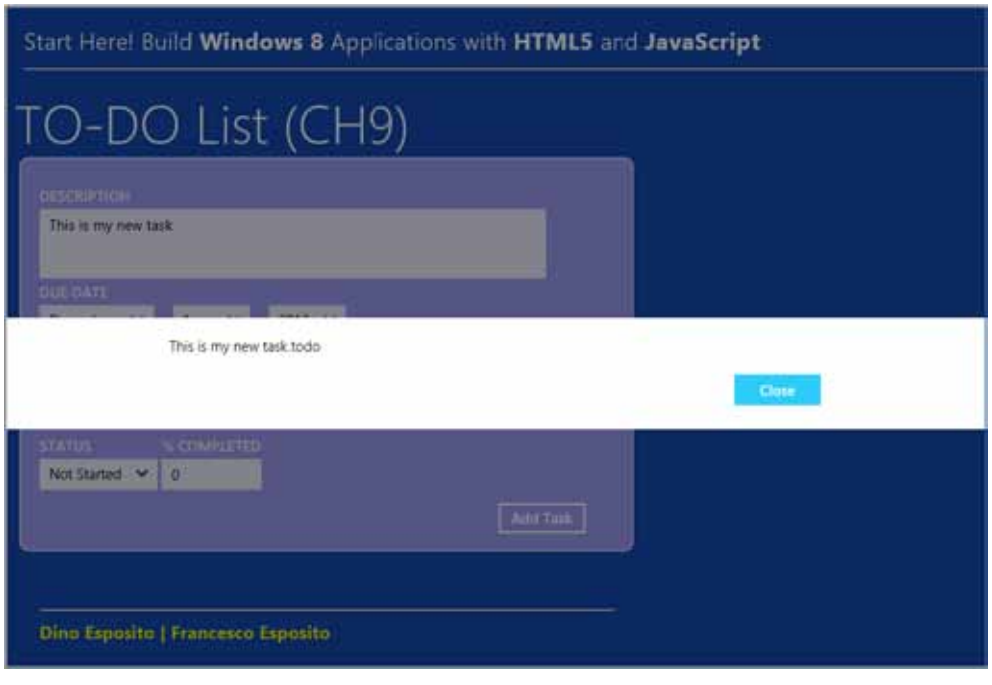

Рис. 9.6. Получение имени выбранного файла

Свойства файла разбиты на три группы: встроенные, основные и расширенные. Встроенными свойствами являются имя и тип файла, они извлекаются вместе с контентом файла. Основные свойства включают в себя размер (`size`) и дату внесения последних изменений (`dateModified`); основные свойства получают путем вызова функции `getBasicPropertiesAsync`. К дополнительным свойствам нужно обращаться посредством предварительного вызова функции `retrievePropertiesAsync`. Рассмотрим пример:

```
file
    .properties
    .retrievePropertiesAsync([fileOwnerProperty, dateAccessedProperty])
    .done(function (extraProperties) {
      TodoList.alert(extraProperties[dateAccessedProperty]);
  }
```

Обычно подобным образом извлекаются такие свойства, как сведения о владельце файла (`fileOwner`) и о дате последнего доступа к файлу (`dateAccessed`).

Выбор файла для загрузки данных

Когда у вас есть сохраненные на диске файлы, рано или поздно вам потребуется их прочитать. Но перед тем как получить возможность это сделать, вам нужно будет выбрать эти файлы в хранилище. Для этого служит компонент выбора открываемого файла. Как уже упоминалось, в качестве средства выбора открываемого файла может использоваться любое приложение, реализующее соответствующий контракт. Но чаще всего вам придется работать с применяемым по умолчанию системным компонентом выбора открываемого файла, а не предлагать другим приложениям собственный компонент. Поэтому давайте узнаем, как работать с компонентом выбора открываемого файла, применяемым по умолчанию.

Использование средства выбора открываемого файла

Средство выбора открываемого файла является экземпляром компонента `FileOpenPicker`. Подобно компоненту `FileSavePicker`, он также имеет свойство `suggestedStartLocation`, позволяющее предложить предпочтительное место, с которого пользователь может начать поиск файла. В следующем коде показано, как настроить этот компонент для выбора графических файлов:

```
var openPicker = new Windows.Storage.Pickers.FileOpenPicker();
  openPicker.suggestedStartLocation =
    Windows.Storage.Pickers.PickerLocationId.picturesLibrary;
openPicker.fileTypeFilter.replaceAll([".png", ".jpg", ".jpeg"]);
```

Функция `replaceAll` указывает на файлы, представляющие для вас интерес при просмотре списка выбора. Список допустимых файловых расширений передается в виде массива. Вы можете также настроить режим просмотра и указать, что вам нужно, чтобы файлы выводились на экран в виде миниатюр:

```
openPicker.viewMode = Windows.Storage.Pickers.PickerViewMode.thumbnail;
```

И наконец, для вывода на экран пользовательского интерфейса выбора и для предоставления пользователю имени выбранного файла нужна функция `pickSingleFileAsync`:

```
openPicker.pickSingleFileAsync().then(function (file) {
    if (file) {
      TodoList.alert("You picked: " + file.name);
    }
});
```

Пользовательский интерфейс компонента выбора открываемого файла показан на рис. 9.7. В нем выводится контент библиотеки Pictures (Изображения), имеющейся

на данном компьютере. Как только пользователь щелкнет на ссылке Pictures, чтобы увидеть дополнительные места поиска файлов, раскроется меню. Это меню содержит перечень мест, предлагаемых по умолчанию системой Windows 8, а также все зарегистрированные нестандартные компоненты выбора файлов.

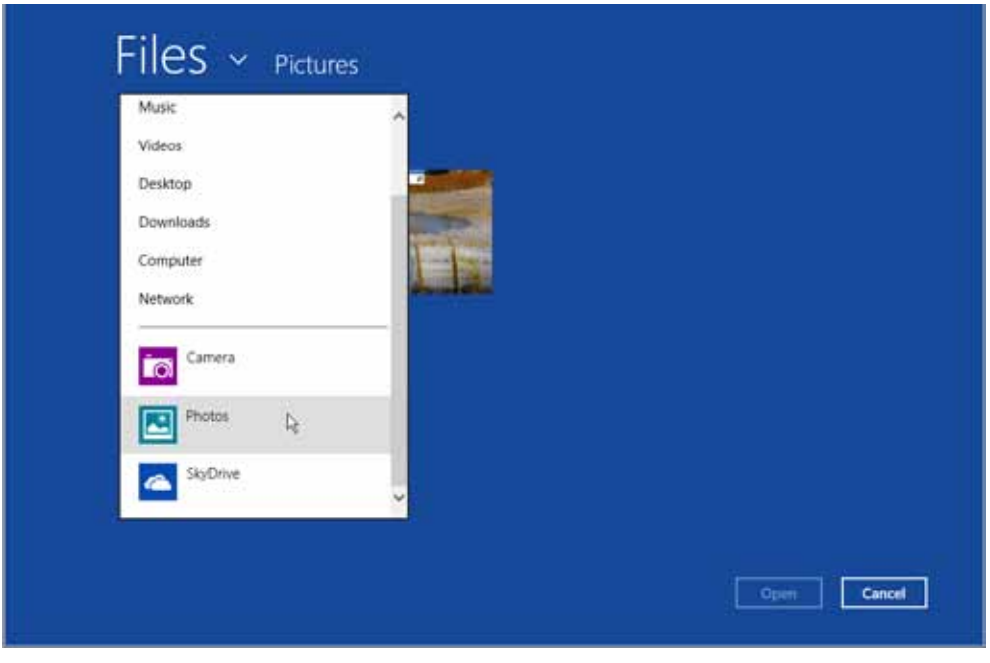

Рис. 9.7. Компонент выбора открываемого файла в действии

По умолчанию вы найдете там нестандартный компонент выбора фотографий `Photos`, предназначенный для получения изображений непосредственно с вебкамеры, а также компонент `SkyDrive`, позволяющий выбрать файл в облаке. Все это — специальные компоненты выбора файлов, которые реализуют упомянутый ранее контракт выбора открываемого файла.

Множественный выбор

В предыдущем примере пользователь мог выбрать только один файл. Чтобы разрешить множественный выбор, нужно внести изменения в функцию, вызывающую пользовательский интерфейс:

```
openPicker.pickMultipleFilesAsync().then(function (files) {
   if (files.size > 0) {
     var buffer = "You picked:\n";
     for (var i = 0; i < files.size; i++) {
       buffer = outputString + files[i].name + "\n";
     }
```

продолжение ⇩

```
      TodoList.alert(buffer);
    } else {
    // Пользователь отклонил диалог выбора, не выбрав ни одного файла
    }
});
```

Функция `pickMultipleFilesAsync` передает следующей функции обратного вызова (коду, переданному в функцию `then`) список файлов, которые мог выбрать пользователь. Доступ ко всем файлам можно получить, запустив цикл с перебором элементов этого списка.

Учтите, что у приложения имеется доступ по чтению и записи к любым файлам, имеющим отношение к компонентам выбора сохраняемых/открываемых файлов.

Выбор папки

Еще один весьма распространенный сценарий касается ситуации, в которой вашему приложению нужно, чтобы пользователь выбрал папку, а не файл.

В Windows 8 для этой цели имеется очень удобный объект — `FolderPicker`. Этот объект используется во многом точно так же, как и все другие средства выбора:

```
// Создание объекта выбора и настройка параметров
 var folderPicker = new Windows.Storage.Pickers.FolderPicker();
 folderPicker.fileTypeFilter.replaceAll(["*"]);
 folderPicker.suggestedStartLocation =
  Windows.Storage.Pickers.PickerLocationId.desktop;
```

Для вывода на экран пользовательского интерфейса нужен следующий код:

```
folderPicker.pickSingleFolderAsync().then(function (folder) {
    if (folder) {
      TodoList.alert(folder.name);
    }
});
```

Учтите, что приложение получает доступ по чтению и записи не только к выбранной папке, но и ко всем вложенным в нее папкам.

Контракт обмена данными

Если у вас есть практика работы с Windows, то вы должны знать о наличии буфера обмена Windows. Это системный механизм, позволяющий пользователю скопировать данные из одного приложения (обычно с помощью комбинации клавиш Ctrl+C) и вставить их в другое приложение (обычно с помощью комбинации клавиш Ctrl+V). Буфер обмена поддерживает множество форматов — данные могут быть скопированы в виде простого текста, в виде текста с оформлением, в виде двоичного образа, но кроме того, поддерживаются специальные форматы того или иного приложения.

По сути, буфер обмена является механизмом, который при ориентации на пользователя поддерживается программным интерфейсом во всех версиях Windows. У приложений Магазина Windows нет доступа к буферу обмена, но это не означает, что различные приложения не могут связываться друг с другом для обмена данными. Просто вместо буфера обмена пользователь может задействовать меню Share (Обмен данными) на выдвигающейся панели.

Публикация данных приложения

В этом упражнении мы научимся делать некоторые данные приложения потенциально доступными любым другим приложениям, зарегистрировавшимся для получения общих данных. Упражнение заключается в разработке такого расширения для приложения TodoList, которое позволило бы ему работать в качестве источника общих данных.

Выбор формата данных

Чтобы стать поставщиком общих данных, не потребуется вносить в приложение какой-то особый код. Достаточно будет специальных компонентов, таких как объекты `DataPackage` и `DataTransferManager`. Первый из них определяет пакет с данными, предназначенными для совместного использования, а второй передает пакет приложениям, запрашивающим эти данные.

При совместном использовании данных основным аспектом, требующим рассмотрения, является формат данных, будь то обычный текст, HTML-разметка или, возможно, двоичный образ. В идеале, чем больше форматов вы поддерживаете, тем шире возможности вашего приложения по совместному использованию данных. Однако в конечном итоге лучшим выбором будет поддержка форматов, наиболее актуальных для конкретного приложения. Обычно это только один или два формата. Поддерживаемые форматы данных перечислены в табл. 9.4.

Таблица 9.4. Поддерживаемые форматы данных для совместного использования

Формат данных	Метод	Описание
Обычный текст	setText	Общие данные состоят из обычной тестовой строки
URI	setUri	Общие данные состоят из URL-адреса, который получатели могут визуализировать в виде объекта, реагирующего на щелчок
HTML	setHtmlFormat	Общие данные состоят из HTML-разметки, включая стили, сценарий и изображения
Текст с оформлением	setRtf	Общие данные состоят из текста в формате RTF (например, из данных, полученных от Microsoft Word)

продолжение ⇩

Таблица 9.4 (*продолжение*)

Формат данных	Метод	Описание
Двоичный образ	setBitmap	Общие данные состоят из изображения, находящегося в памяти
Файлы	setStorageItems	Общие данные состоят из файлов

Пакет данных

В столбце «Метод» табл. 9.4 дается ссылка на методы, доступные в объекте `DataPackage` и позволяющие вести обмен данными в конкретном формате. Создание допустимого пакета данных является основной обязанностью приложения, являющегося источником общих данных. Объект `DataPackage` может содержать данные в одном или нескольких форматах, перечисленных в табл. 9.4. Получение наиболее актуальных данных возлагается на приложение, которое запрашивает эти данные.

Например, исходное приложение Mail, имеющееся в Windows 8, работает в качестве приемника общих данных. Оно способно получать данные в различных форматах и внедрять данные в тело почтового сообщения. Если приложение передает данные в виде простого текста и в виде HTML, то формат HTML выбирается как более предпочтительный. Вскоре вам удастся поэкспериментировать с подобным поведением.

Наделение приложения TodoList способностями источника общих данных

В качестве первого шага на пути обмена данными с другими приложениями вам нужно зарегистрировать в приложении обработчик события `datarequested`. В жизненном цикле приложения это нужно сделать как можно скорее.

Обработка запросов данных

Откройте файл todolist.js и добавьте в конец метода `TodoList.init` следующий код:

```
// Инициализация контракта источника общих данных
 var clipboard = Windows.ApplicationModel.DataTransfer.DataTransferManager.
 getForCurrentView();
 clipboard.addEventListener("datarequested", function (e) {
   // Получение информации для совместного использования
   var currentTask = TodoList.getTaskFromUI();

   // Обмен информацией в виде обычного текста
   TodoList.shareDataAsPlainText(e, currentTask);

   // Обмен информацией в формате HTML
```

```
    TodoList.shareDataAsHtml(e, currentTask);
});
```

Переменная `clipboard` ссылается на объект диспетчера передачи данных, который активен для текущего окна. Диспетчер передачи данных является системным компонентом, отвечающим за передачу данных в приложение и из приложения. Метод `addEventListener` служит для регистрации обработчика события `datarequested`. Это событие инициализируется, когда пользователь выводит панель Share (Обмен данными) и хотя бы одно приложение готово принять данные.

ПРИМЕЧАНИЕ

Windows 8 оставляет доступной кнопку Share (Обмен данными) на выдвигающейся панели, даже если текущее приложение не поддерживает контракт источника общих данных. Поэтому пользователь может всегда попытаться реализовать обмен контентом из любого приложения. Однако если объект диспетчера передачи данных (DataTransferManager) обнаружит, что текущее приложение не предоставило обработчик события datarequested, появится сообщение, показанное на рис. 9.8.

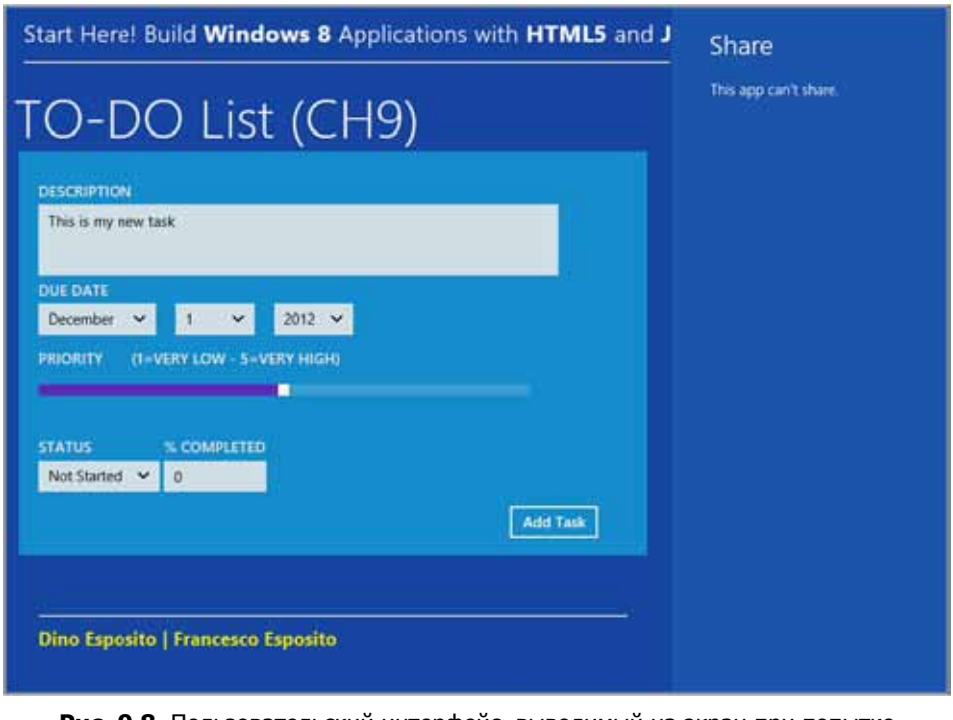

Рис. 9.8. Пользовательский интерфейс, выводимый на экран при попытке осуществить обмен данными из приложения, не поддерживающего контракт источника общих данных

В обработчике события datarequested осуществляются сбор данных для обмена, нужное форматирование данных и их пакетирование в объекте DataPackage. В показанном фрагменте кода данные, предназначенные для совместного использования, представлены созданным заданием. Обмен информацией о задании осуществляется в формате простого текста и в формате HTML. Порядок, в котором данные добавляются в пакет (в предыдущем фрагменте кода первым добавляется простой текст), роли не играет. Формат обмена зависит от параметров приложения-приемника.

Обмен данными в виде простого текста

Теперь нужно в конец файла todolist.js добавить новую функцию, отвечающую за обмен информацией о задании с другими приложениями:

```
TodoList.shareDataAsPlainText = function (e, task) {
   var request = e.request;

   // Добавление данных в пакет
   request.data.properties.title = "TO DO";
   request.data.properties.description = task.description;
   request.data.setText(task.description + " due by "
                              + task.dueDate.toLocaleDateString());
}
```

Пакет для данных предоставляется объектом request.data. У компонента DataPackage имеется два типовых свойства: title (название) и description (описание). Для них можно установить значения, чтобы дать описание тех данных, которые вы собираетесь предоставить, в расчете на то, что ими может воспользоваться как панель Share (Обмен данными), так и целевое приложение. Поэтому установка для них осмысленных значений всегда приветствуется. В данном случае свойству title присваивается значение в виде статического текста «TO DO», а свойству description — описание конкретной задачи.

И наконец, подготавливается та строка теста, которая и будет предметом обмена. Она копируется в пакет методом setText. На рис. 9.9 показан эффект от внесенного изменения: теперь приложение TodoList может передавать данные приложению Mail через элемент Share (Обмен данными) на выдвигающейся панели.

Как только пользователь щелкнет на одном из перечисленных приложений (на показанной копии экрана в качестве получателя фигурирует только приложение Mail), управление будет передано выбранному приложению. Приложение проверяет пакет данных, убеждается в том, что он содержит данные, которые оно в состоянии обработать, извлекает данные в наиболее удобном для себя формате, а затем использует эти данные. На рис. 9.10 показано, что может сделать приложение Mail с переданными ему данными: оно подготавливает новое электронное сообщение и автоматически вводит в тело этого сообщения переданный текст.

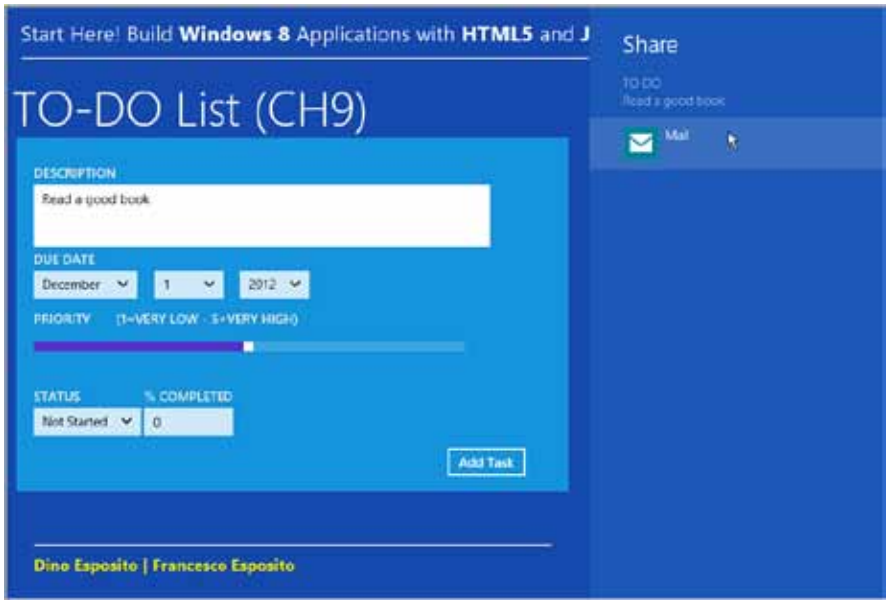

Рис. 9.9. Приложение TodoList готово к совместному использованию информации о задании

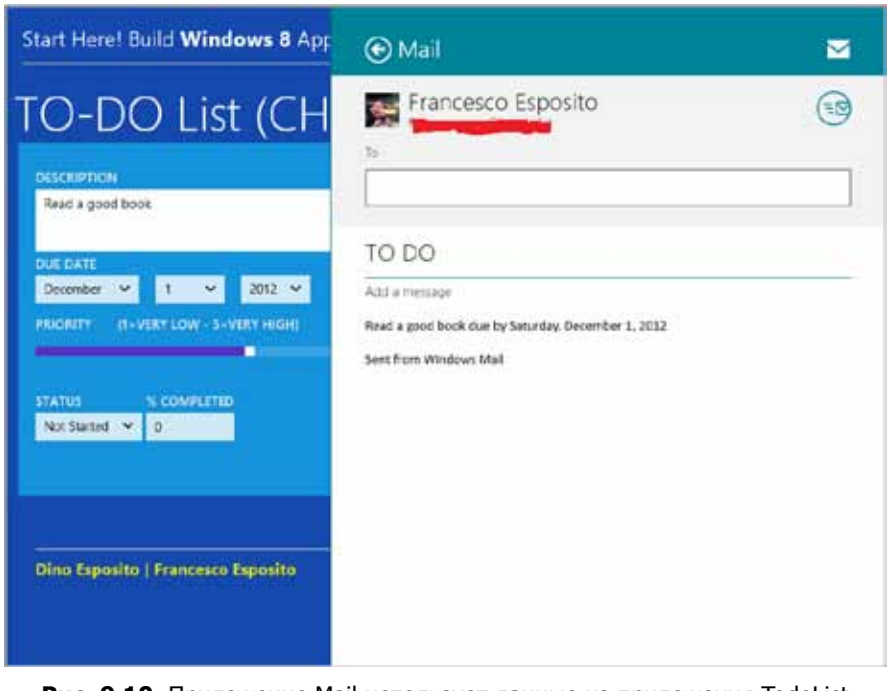

Рис. 9.10. Приложение Mail использует данные из приложения TodoList

Обмен данными в формате HTML

Обмен данными в формате HTML осуществляется немного сложнее. Наиболее интересным является способ форматирования HTML-кода. Можно создать HTML-строку программным способом, объединив текст и HTML-элементы, или же можно записать HTML-код непосредственно на HTML-страницу и прочитать его из нее. Преимущество последнего способа заключается в том, что ваш код будет легче понять, а с HTML-структурой легче разобраться и ее проще модифицировать. Кроме того, для редактирования подойдет любой экранный редактор.

Откройте файл default.html и добавьте в конце файла непосредственно перед закрывающим элементом body следующую разметку:

```
<div id="shareHtml" style="display:none;">
   <img style="float:left;" src="ms-appx:///images/todolist-icon.png" alt="" />
   <div style="padding:3px;background:#999;color:#fff">DESCRIPTION:</div>
   <div><span data-win-bind="innerText: description" /></div>
   <div style="padding:3px;background:#999;color:#fff">DUE DATE:</div>
   <div><span data-win-bind="innerText: dueDate TodoList.dateForDisplay" /></div>
   <div style="padding:3px;background:#999;color:#fff">PRIORITY:</div>
   <div><span data-win-bind="innerText: priority" /></div>
   <div style="padding:3px;background:#999;color:#fff">STATUS:</div>
   <div><span data-win-bind="innerText: status" /></div>
   <div style="padding:3px;background:#999;color:#fff">COMPLETED:</div>
   <div><span data-win-bind="innerText: percCompleted" /></div>
</div>
```

Именно эту HTML-разметку приложение выставит на обмен. HTML-разметка заполняется данными о задании посредством привязки данных — аналогичный подход использовался ранее для создания сводки о задании. Следует также заметить, что, поскольку вы не собираетесь показывать этот HTML-фрагмент на странице, важно явным образом установить для CSS-атрибута `display` корневого элемента значение none. Тем самым гарантируется невидимость HTML-блока.

Теперь добавьте в конце файла todolist.js следующую функцию:

```
TodoList.shareDataAsHtml = function (e, task) {
   var request = e.request;
   request.data.properties.title = "TO DO";
   request.data.properties.description = task.description;

   // Загрузка HTML-разметки и запуск механизма привязки данных
   var elem = document.getElementById("shareHtml");
   WinJS.Binding.processAll(elem, task);
   var rawHtml = elem.innerHTML;

   // Приведение исходного HTML-кода в соответствие требованиям Windows 8
   var html = Windows.ApplicationModel.DataTransfer.HtmlFormatHelper.
   createHtmlFormat(rawHtml);
   request.data.setHtmlFormat(html);

   // Следующая дополнительная работа нужна, ТОЛЬКО если HTML-код
   // ссылается на изображение.

   // Это URL-адрес изображения, находящийся в HTML-блоке.
   // Он преобразуется в URI-объект и сохраняется в качестве
```

```
    // потока в памяти в ресурсах DataPackage.
    var localImage = "ms-appx:///images/todolist-icon.png";
    var url = new Windows.Foundation.Uri(localImage);
    var streamRef = Windows.Storage.Streams.RandomAccessStreamReference.
                        createFromUri(url);
    request.data.resourceMap[localImage] = streamRef;
}
```

HTML-блок извлекается в виде строки путем чтения свойства `innerHTML` элемента `div`. Исходный HTML-код должен быть отформатирован в соответствии с требованиями Windows 8. Это форматирование выполняется вспомогательным методом `createHtmlFormat`. В завершение для упаковки HTML-описания задания вызывается функция `setHtmlFormat`.

Если в HTML содержится ссылка на одно или несколько изображений, потребуются дополнительные усилия. В частности, ссылки на изображения можно делать с помощью специального протокола ms-appx. Этот протокол определяет изображения как собственную часть приложения. Иными словами, если нужно включить изображение, принадлежащее ресурсам приложения, следует использовать протокол ms-appx. Если есть намерение сослаться на изображения с удаленных URL-адресов, то для этого вполне подойдет протокол HTTP. В показанном примере файл todolist-icon.png является изображением, хранящимся в папке images проекта приложения.

На рис. 9.11 показано электронное сообщение, используемое для совместного использования задания.

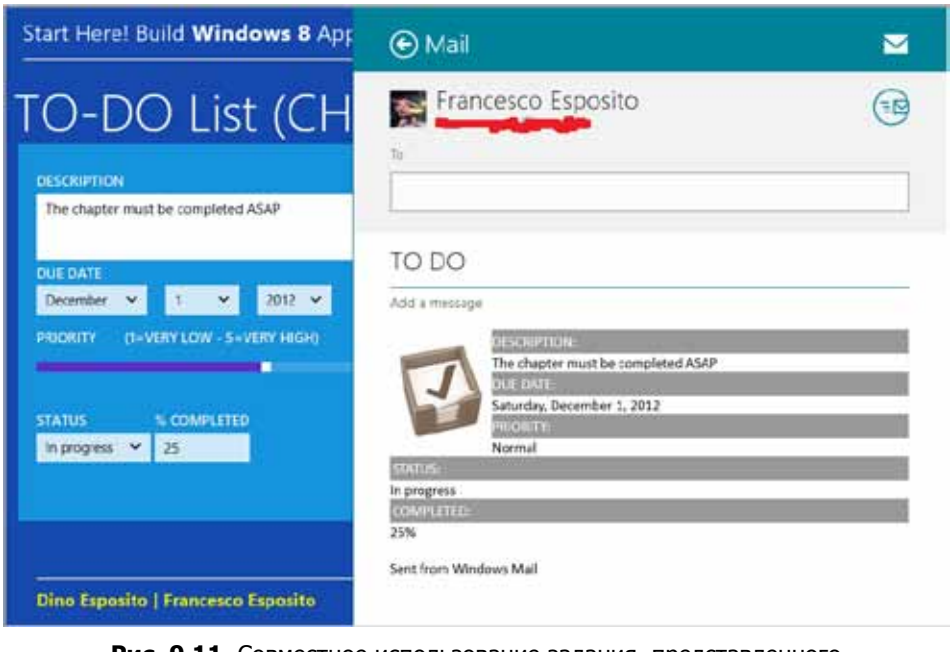

Рис. 9.11. Совместное использование задания, представленного в формате HTML

Условный обмен данными

Иногда приложение, являющееся источником данных для совместного использования, не может их предоставить. В качестве типичного примера можно привести ситуацию, когда данные, предлагаемые для обмена, должен выбрать пользователь, и, если он этого не сделает, данных для обмена не будет. Так, от приложения Todo-List вполне резонно потребовать, чтобы совместному использованию подлежали только те данные, у которых имеется готовое описание.

Условный обмен данными реализуется в обработчике события `datarequested`. Код подключения контракта источника общих данных, ранее добавленный к методу `TodoList.init`, нужно модифицировать следующим образом:

```
TodoList.clipboard.addEventListener("datarequested", function (e) {
   var currentTask = TodoList.getTaskFromUI();

   if (currentTask.description.length === 0) {
     e.request.failWithDisplayText("Indicate a description of the task.");
     return;
   }

   TodoList.shareDataAsHtml(e, currentTask);
   TodoList.shareDataAsPlainText(e, currentTask);
});
```

Метод `failWithDisplayText` приводит к срыву операции по обмену данными. Пользователю может выводиться текст, описывающий ситуацию. Если разъяснительный текст не предоставляется, пользователь получит типовое сообщение, утверждающее, что в данном случае приложение не может поделиться своими данными.

Программируемый обмен данными

Выдвигающаяся панель — не единственное средство доступа к панели Share (Обмен данными). Каждое приложение может предложить пользователю собственный пользовательский интерфейс для обмена данными. Например, можно добавить к файлу default.html новую кнопку, поместив ее рядом с уже существующей кнопкой Add Task (Добавить задание):

```
<button id="buttonShare">Share</button>
```

В файл TodoList.init также нужно добавить обработчик щелчка на этой кнопке:

```
document.getElementById("buttonShare").addEventListener(
               "click", TodoList.shareClick);
```

И наконец, трюк по программной визуализации панели Share (Обмен данными) без выдвигающейся панели реализуется с помощью новой функции `TodoList.shareClick`:

```
TodoList.shareClick = function () {
   Windows.ApplicationModel.DataTransfer.DataTransferManager.showShareUI();
}
```

ПРИМЕЧАНИЕ

Чтобы попасть в перечень целевых приложений для обмена данными (таких, как приложение Mail, с которыми мы работали в данном упражнении), приложению нужно реализовать контракт приемника общих данных. Этот контракт реализуется путем добавления в проект специального объекта из диалогового окна Add New Item (Добавить новый объект) Microsoft Visual Studio. Добавление нового приемника общих данных приведет к появлению в вашем проекте трех новых файлов и необходимости внесения некоторых изменений в манифест проекта. Вы получите новые CSS-, HTML- и JavaScript-файлы, которые, соответственно, предоставят для новой страницы стиль, разметку и код. Изменения, вносимые в манифест, проинформируют Windows 8 о том, что ваше приложение намеревается участвовать в контракте обмена данными. Когда пользователь выбирает ваше приложение для получения какого-то совместно используемого контента, на экран выводится новая страница, чья логика позволяет вам получить и обработать данные. От приложения, выполняющего контракт приемника общих данных, не следует ожидать вывода полноценного пользовательского интерфейса. Достаточно минимального пользовательского интерфейса для решения конкретной задачи, касающейся полученного контента. Пример приложения, поддерживающего контракт приемника общих данных, можно найти в Windows SDK.

Предоставление страницы настройки

Панель Settings (Настройки) на выдвигающейся панели должна быть тем местом, где пользователь может быстро получить доступ к параметрам приложения. Приложение Магазина Windows может предоставить пользователю набор дополнительных страниц, которые можно просматривать в панели Settings (Настройки) с целью изменения параметров, влияющих на поведение приложения, а также страниц Help (Справка) и About (О программе). В данном упражнении для приложения TodoList мы создадим страницу Settings (Настройки).

Наполнение панели Settings

Когда пользователь щелкает на панели Settings (Настройки), расположенной на выдвигающейся панели, приложение получает оповещение о событии `onsettings`. Поэтому первым шагом на пути решения поставленной задачи станет реализация обработчика этого события.

Создание всплывающего компонента для настройки

Панель Settings (Настройки) является, по сути, всплывающим компонентом, в котором настройка выполняется с помощью перечня прикладных команд. Прикладная команда представляет собой HTML-страницу и строку названия. Откройте файл default.html и добавьте в нее перед вызовом метода `app.start` следующий код:

```
app.onsettings = function (e) {
    e.detail.applicationcommands = {
      "about": {
        href: "/pages/about.html",
        title: "About"
      },
      "privacy": {
        href: "/pages/privacy.html",
        title: "Privacy"
      },
      "settings": {
        href: "/pages/settings.html",
        title: "Settings"
      }
    }
    WinJS.UI.SettingsFlyout.populateSettings(e);
};
```

Следующим обязательным шагом является создание трех новых HTML-страниц. Все они должны быть созданы в папке pages проекта с упомянутыми ранее названиями: about.html, privacy.html и settings.html. Это делается с помощью команды Add New Item (Добавить новый элемент) контекстного меню окна проекта в Visual Studio. Пока не будем заниматься нужным контентом, а просто оставим исходную разметку, включенную системой Visual Studio в каждую из созданных HTML-страниц.

Как показано на рис. 9.12, на панели Settings (Настройки) перечисляются три дополнительные позиции, по одной для каждой зарегистрированной команды. Но если щелкнуть на любой из них, ничего не произойдет. Поэтому настало время внести ряд изменений в HTML-код созданных страниц.

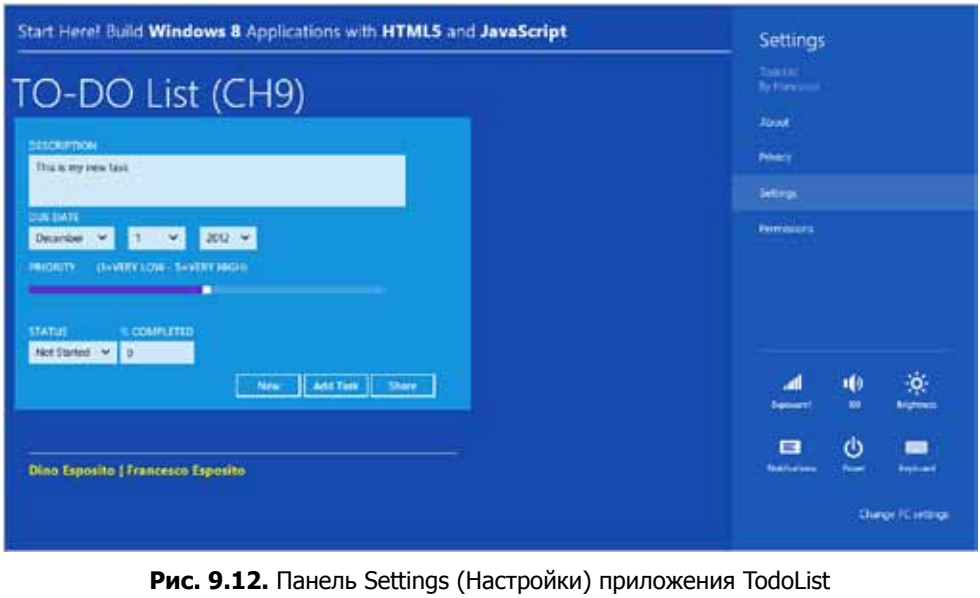

Рис. 9.12. Панель Settings (Настройки) приложения TodoList

Создание страницы только для чтения

В отличие от известного исключения, касающегося страницы, используемой для конфигурирования приложения, большинство упомянутых здесь страниц являются страницами, предназначенными только для чтения. Они просто предоставляют пользователям такую информацию, как сведения о выпуске, соглашение для конечного пользователя, справочные сведения или, возможно, информация об авторе. У всех этих страниц может быть одинаковый макет, но разный контент. Давайте, к примеру, обратимся к странице About (О программе).

Откройте файл about.html и введите в тег body следующую разметку:

```
<div id="aboutContainer"
   data-win-control="WinJS.UI.SettingsFlyout"
   data-win-options="{settingsCommandId:'about'}">
   <div class="win-ui-dark win-header">
     <button type="button"
       onclick="WinJS.UI.SettingsFlyout.show()"
       class="win-backbutton"></button>
     <div class="win-label">About TodoList</div>
   </div>
   <div class="win-content">
     <div class="win-settings-section">
       <!-- Сюда нужно поместить контент -->
       <h1>v1.0.0.0.1</h1>
     </div>
   </div>
</div>
```

Важно, чтобы в странице была реализована привязка корневого элемента `div` к компоненту `SettingsFlyout`. Каким будет значение атрибута `id` элемента `div`, особой роли не играет, но внутри приложения оно должно быть уникальным. Также уникальным должно быть значение идентификатора команды, присваиваемой атрибуту `data-win-options`.

Для придания странице внешнего вида, согласующегося с внешним видом других приложений, можно добавить дочерний элемент `div`, стиль которого определяется CSS-классом `win-header`. Кроме того, свои графические предпочтения можно выразить с помощью класса `win-ui-dark`. У вас есть выбор между классами `win-ui-dark` и `win-ui-light`. И наконец, вам нужна кнопка для возвращения в приложение. Поэтому в показанный код включен элемент `button`. В завершение для элемента `div` задан CSS-класс `win-label`, определяющий внешний вид заголовка страницы.

Весь контент страницы помещается в элемент `div`, стиль которого определяется классом `win-content`. Выбор структуры и стиля этого контента полностью отдается на ваше усмотрение. Это исключительно вопрос стиля, никакого влияния на поведение приложения он не оказывает.

Внешний вид страницы About (О программе) показан на рис. 9.13.

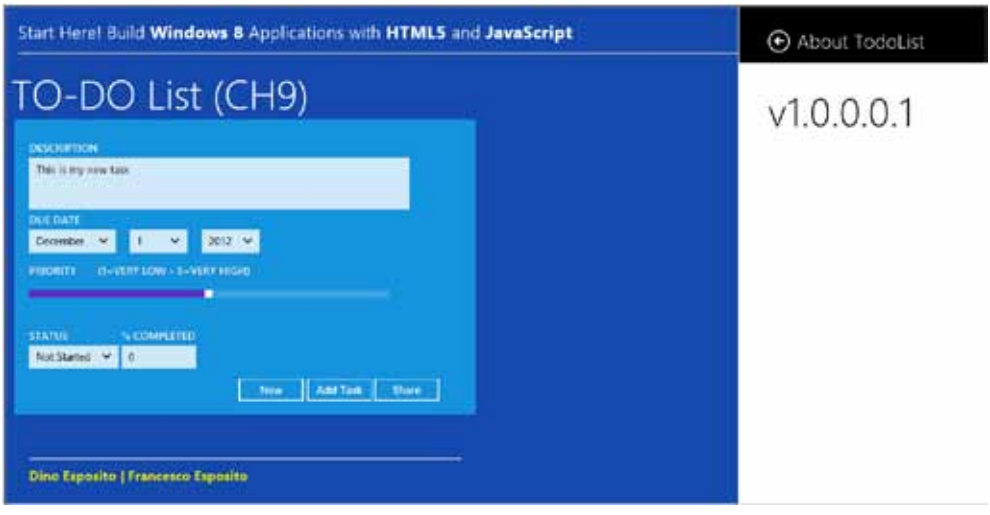

Рис. 9.13. Страница About (О программе)

Все рассмотренные здесь этапы создания страницы About (О программе) можно легко повторить для создания любой другой страницы, которая должна появиться в перечне панели Settings (Настройки) и которая не требует взаимодействия с пользователем.

ПРИМЕЧАНИЕ

Пользователь может убрать панель Settings (Настройки), щелкнув за ее пределами или коснувшись такой области. Можно также щелкнуть на кнопке Back (Назад) или коснуться этой кнопки.

Создание функциональной страницы Settings

Макет страницы Settings (Настройки), то есть страницы, предлагаемой пользователю для изменения параметров и конфигурирования приложения для разных режимов работы, не отличается от макета страниц, предназначенных только для чтения. Тем не менее создать страницу Settings (Настройки) сложнее, поскольку для этого требуется где-то сохранять варианты настройки и использовать их во всем приложении. Поэтому в предыдущий код придется внести существенные изменения.

Определение объекта настройки всего приложения

Если предполагается, что пользователи вашего приложения должны иметь возможность задавать его параметры, у вас должен быть объект, определяющий все варианты настройки. Откройте файл todolist.js и добавьте следующий код:

```
var TodoListSettings = WinJS.Class.define(function () {
   var localSettings = Windows.Storage.ApplicationData.current.localSettings;
   var that = {};
   that.defaultPriority = TodoList.Priority.Normal;
   that.load = function () {
     _loadFromSettings();
   };
   that.save = function () {
     _saveToSettings();
   };

   function _loadFromSettings() {
     var priority = localSettings.values["defaultPriority"];
     if (priority) {
       that.defaultPriority = priority;
     }
   };

   function _saveToSettings() {
     localSettings.values["defaultPriority"] = that.defaultPriority;
   };

   return that;
 });

TodoList.settings = new TodoListSettings();
```

ВНИМАНИЕ

Конкретное местоположение этого кода при условии, что за ним следует определение объекта TodoList.Priority, особого значения не имеет. Тем не менее рекомендуется добавить этот код в верхнюю часть файла, сразу же после определения объекта TodoList. Затем должно следовать определение объекта TodoList.Priority.

У объекта `TodoListSettings` есть только одно свойство — `defaultPriority`. Дело в том, что в данном примере нужен только один настраиваемый параметр — предлагаемый по умолчанию уровень приоритета для любого вновь создаваемого задания. До сих пор этот уровень по умолчанию получал значение `Normal`, но теперь мы дали пользователю возможность его задать.

Однако объект `TodoListSettings` не ограничивается определением свойства `defaultPriority`, у него есть также два метода, отвечающие за загрузку и сохранение. Замысел заключается в том, чтобы объект `TodoListSettings` сам отвечал за загрузку и сохранение своего контента, используя постоянное хранилище и не требуя простых вызовов методов загрузки или сохранения. Детали того, как параметры сохраняются или извлекаются, за пределами этого объекта не известны.

Кроме того, экземпляр объекта `TodoListSettings` создается в процессе инициализации приложения и становится доступным через объект `TodoList.settings`. В результате приложение загружает свои параметры из постоянного хранилища при запуске и к этим параметрам имеется глобальный доступ в течение всего жизненного цикла приложения.

Надежное сохранение параметров приложений

Windows 8 предоставляет приложения с системным словарем, где данные могут сохраняться в виде пар, состоящих из уникального идентификатора и соответствующего ему значения. Значения могут быть представлены только самыми простыми типами, например строками и целыми числами. Системным объектом, который предоставляет доступ к этому хранилищу данных, является `Windows.Storage.ApplicationData.current.localSettings`. Этот объект предоставляет доступ к свойству `values`, являющемуся тем самым словарем, в котором хранятся данные.

Например, предлагаемый по умолчанию уровень приоритета, установленный для приложения, хранится в записи, идентификатор которой соответствует названию свойства:

```
var priority = localSettings.values["defaultPriority"];
```

Следует заметить, что это всего лишь рекомендуемый прием, и вы можете называть запись в словаре как угодно. Чтобы значения сохранялись на постоянной основе, нужно воспользоваться следующим кодом:

```
localSettings.values["defaultPriority"] = someValue;
```

Для завершения инициализации приложения требуется вызов, призванный загружать параметры при запуске приложения:

```
TodoList.settings.load();
```

Эту строку кода нужно добавить в конец метода `TodoList.init` в файле todolist.js. После включения параметров в приложение их нужно использовать там, где это имеет наибольший смысл. А это требует внесения дальнейших изменений в файл todolist.js.

В частности, нужно отредактировать объект `Task`, чтобы он устанавливал для свойства приоритета предлагаемое по умолчанию значение, считанное из параметров:

```
var Task = WinJS.Class.define(function () {
   var that = {};
   that.description = "This is my new task";
   that.dueDate = TodoList.firstOfNextMonth();
   that.priority = TodoList.Priority.Normal;
   ...
   if (TodoList.settings.defaultPriority != "undefined")
     that.priority = TodoList.settings.defaultPriority;
   return that;
});
```

Еще одно изменение касается метода `TodoList.performInitialBinding`. На данный момент вам, возможно, захочется инициализировать пользовательский интерфейс на основе полученного объекта `Task`, а не того объекта `Task`, который создается внутри приложения:

```
TodoList.performInitialBinding = function (task) {
   // var task = new Task();     // Эту строку нужно убрать

   // Сюда помещается весь остальной код
   ...
}
```

И наконец, нужно ввести новую функцию:

```
TodoList.displayTask = function (task) {
   TodoList.performInitialBinding(task);
}
```

Теперь вызов новой функции `TodoList.displayTask` нужно поместить в самый конец метода `TodoList.init`. В то же время нужно удалить все вызовы `TodoList.performInitialBinding`, которые могут иметься в методе `TodoList.init`. Новый код метода `TodoList.init` должен иметь следующий вид:

```
TodoList.init = function () {
   // Регистрация обработчика события изменения размеров
   window.onresize = addEventListener('resize', TodoList.onResize, false);

   // Регистрация обработчиков кнопок
   document.getElementById("buttonAddTask").addEventListener(
     "click", TodoList.addTaskClick);
   document.getElementById("buttonShare").addEventListener(
     "click", TodoList.shareClick);

   // Инициализация контракта источника общих данных
   var view = Windows.ApplicationModel.DataTransfer.
     DataTransferManager.getForCurrentView();
   view.addEventListener("datarequested", function (e) {
     var currentTask = TodoList.getTaskFromUI();
     if (currentTask.description.length === 0) {
       e.request.failWithDisplayText("Indicate a description of the task.");
       return;
     }
     TodoList.shareDataAsHtml(e, currentTask);
     TodoList.shareDataAsPlainText(e, currentTask);
   });

   // Загрузка параметров и инициализация представления
   TodoList.settings.load();
   TodoList.displayTask(new Task());
}
```

После внесения этих изменений при каждом запуске приложения предоставляемое по умолчанию значение параметра приоритета задания считывается из параметров приложения и применяется для инициализации пользовательского интерфейса.

ПРИМЕЧАНИЕ

В частности, внесенные здесь изменения в метод performInitialBinding и введение новой функции displayTask существенно упростят включение новых функций в приложение в следующих главах.

Создание страницы настройки

Теперь настало время обратить внимание на страницу settings.html. Этой странице назначается тот же макет, что и другим страницам. В элемент body нужно добавить следующую разметку:

```
<div id="settingsContainer"
   data-win-control="WinJS.UI.SettingsFlyout"
   data-win-options="{settingsCommandId:'settings', width:'narrow'}">

   <div class="win-ui-dark win-header">
     <button type="button" onclick="WinJS.UI.SettingsFlyout.show()"
     class="win-backbutton"></button>
     <div class="win-label">TodoList Settings</div>
   </div>
   <div class="win-content">
     <div class="win-settings-section">
       <h3>DEFAULT PRIORITY <br />(1=VERY LOW - 5=VERY HIGH)</h3>
       <input id="taskPriority_settings" type="range" min="1" max="5"
         data-win-bind="value: defaultPriority;" />
     </div>
   </div>
</div>
```

Страница Settings (Настройки) показана на рис. 9.14.

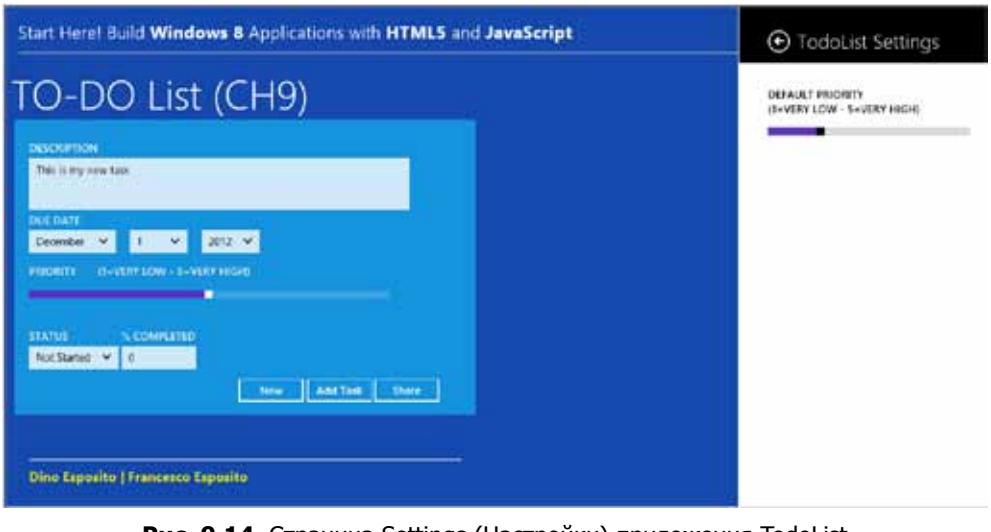

Рис. 9.14. Страница Settings (Настройки) приложения TodoList

Заключительным шагом является инициализация страницы Settings (Настройки) текущими параметрами и сохранение любого нового значения, которое пользователь может выбрать на странице, обратившись к настройке приложения. Чтобы

реализовать задуманное, нужно перехватить два события, относящихся к всплывающей панели настройки. А для этого к проекту требуется добавить новый JavaScript-файл settings.js. Кроме того, со страницы settings.html требуется сослаться на этот файл так же, как и на файл todolist.js:

```
<script src="/js/todolist.js"></script>
<script src="/js/settings.js"></script>
```

В файл settings.js нужно добавить следующий код:

```
(function () {
   "use strict";
   var page = WinJS.UI.Pages.define("/pages/settings.html", {
     ready: function (element, options) {
       document.getElementById("settingsContainer")
               .winControl
               .addEventListener("beforeshow", beforeShow);
       document.getElementById("settingsContainer")
               .winControl
               .addEventListener("afterhide", afterHide);
     }
   });

   function beforeShow() {
     loadSettings();
   };

   function afterHide() {
     saveSettings();
   };

   function loadSettings() {
     var bindableElement = document.getElementById("settingsContainer");
     WinJS.Binding.processAll(bindableElement, TodoList.settings);
   };

   function saveSettings() {
     var priorityElem = document.getElementById("taskPriority_settings");
     var currentPriority = priorityElem.value;
     TodoList.settings.defaultPriority = currentPriority;
     TodoList.settings.save();
   };
})();
```

В этом коде регистрируются обработчики событий `beforeshow` и `afterhide` панели, всплывающей вслед за страницей Settings (Настройки). Благодаря им вы сможете выполнять свой код непосредственно перед выводом на экран панели Settings (Настройки) и сразу же после того, как эта она скрывается.

Излишне говорить, что в обработчике `thebeforeshowevent` вы инициализируете элементы управления на панели Settings (Настройки) текущими параметрами, а в обработчике `afterhideevent` сохраняете изменения, сделанные в параметрах приложения.

ВНИМАНИЕ

Важно заметить, что благодаря усилиям, предпринятым ранее в данном упражнении — изоляции параметров приложения в отдельном объекте, — теперь вы можете считывать и сохранять параметры быстро и просто, не задумываясь о многочисленных деталях, касающихся инфраструктуры.

Результатом изменений стала появившаяся возможность открыть страницу Settings (Настройки), изменить значение приоритета, предлагаемое по умолчанию, и назначать это значение при каждом создании нового задания.

ПРИМЕЧАНИЕ

Если все этапы данного упражнения были выполнены успешно, у вас может появиться вполне резонный вопрос: с какой стати мне было предписано обрабатывать события до и после всплытия окна, когда можно было просто установить на странице настройки кнопку Save (Сохранить)? В руководствах по разработке приложений Магазина Windows настоятельно рекомендуется не использовать кнопки сохранения на страницах настройки и, соответственно, использовать в качестве единственно возможного варианта обработку событий окна до и после его всплытия.

Выводы

В этой довольно длинной главе мы узнали, как лучше интегрировать приложение Магазина Windows в окружающую среду. С технической точки зрения интеграция достигается посредством контрактов и расширений, которые являются также основными инструментами разработчика для настройки и дополнения базовой функциональности Windows.

Вопросы реализации контрактов и написания расширений выходят за рамки темы этой книги, предназначенной для начинающих разработчиков, а нужные примеры можно найти в Windows SDK, а также в более серьезных книгах по программированию в среде Windows 8. Вместо этого можно использовать контракты в качестве сервисов, что значительно проще и требует изучения всего нескольких объектов. В данной главе мы также коснулись вопросов, относящихся к файлам и хранилищам, которые и являются основной темой следующей главы.

10 Сохранение данных приложений

Очень важно уже даже то, что вы просто не сдаетесь.

Стивен Хокинг

Никакие приложения, исключая разве что совсем простые, которые мы разрабатывали в наших первых упражнениях, не могут функционировать без сохранения данных. Когда пользователи работают с приложением, они производят информацию, вводя новые данные или обрабатывая уже известные значения. Эта информация не должна пропадать с завершением работы приложения, а должна где-нибудь храниться, чтобы ее можно было снова загрузить в следующем сеансе работы или по запросу.

Чтобы сохранять данные, программы используют файлы; причем иногда это могут быть особые файлы, известные как *базы данных*. По сути, база данных является большой коллекцией файлов, которой владеет и управляет специальное и вполне конкретное приложение. С точки зрения системных ресурсов взаимодействие с базой данных является более затратной операцией, поскольку в ней используются специальные протоколы для обмена данными между приложениями.

И хотя в этой главе базы данных нам не понадобятся, мы выполним несколько упражнений, в которых используются файлы. В частности, мы доработаем приложение TodoList, чтобы оно могло сохранять задания в файлах и заново загружать их по мере необходимости. В процессе этой работы мы многое узнаем о прикладном

программном интерфейсе (Application Programming Interface, API), предназначенном для работы с сохраняемыми данными в приложениях Магазина Windows.

Сохраняемые объекты приложения

В первом упражнении данной главы мы немного доработаем версию приложения TodoList, разработанную в предыдущей главе. С помощью этого приложения можно будет выбрать файл и сохранить в нем некоторые детали задания.

Превращение объектов Task в сохраняемые объекты

Сделайте копию проекта TodoList в том его состоянии, в котором он был к концу главы 9, и назовите этот новый проект TodoList-Persistence. Чтобы избежать недоразумений, нужно также открыть файл default.html и отредактировать название приложения. Замените в элементе `body` этой страницы элемент `h1` следующим кодом:

```
<h1>TO-DO List (CH10)</h1>
```

В своем текущем состоянии приложение позволяет выбрать имя файла в папке на локальном диске и вернуть объект, представляющий нужный файл. Пока что этого файла нет, поэтому на следующем этапе мы создадим реальный файл и сохраним в нем какие-нибудь реальные данные. Но сначала нужно познакомиться с объектами ввода-вывода (Input/Output, I/O) в Windows 8.

Добавление задания

В главе 9 мы научились вызывать компонент выбора сохраняемого файла, позволяющий выбрать файл в папке на диске. Рисунок 10.1 кратко иллюстрирует порядок действий при добавлении задания.

Сначала пользователь щелкает на кнопке Add Task (Добавить задание), затем ему предоставляется сводка со сведениями о текущем задании, и, если его все устраивает, он продолжает операцию, сохраняя задание на диске. Приложение вызывает средство выбора файла, и пользователь выбирает нужную папку, после чего вводит имя файла. Как вы помните по главе 9, в методе `TodoList.invokeSavePicker` из файла todolist.js у нас есть следующий код:

```
savePicker.pickSaveFileAsync().then(function (file) {
 if (file) {
   TodoList.alert(file.name);
 }
});
```

Теперь пора заменить этот код, вызывающий обычное окно сообщения, кодом, реально создающим файл.

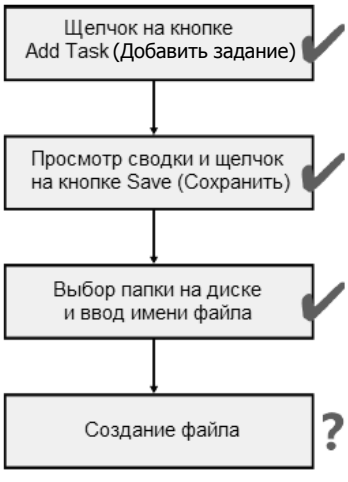

Рис. 10.1. Порядок действий при добавлении задания

Сохранение данных в файле

Пользы от компонента выбора сохраняемого файла, предлагаемого Microsoft Windows 8, намного больше, чем можно было бы ожидать. Он возвращает не просто объект с описанием файла, который вы намереваетесь создать, а самый настоящий файловый объект. Иными словами, объект, получаемый от этого компонента, ссылается на файл, который уже создан для вас. То есть реально существующий файл, хотя и пустой. Следовательно, вам остается лишь записать в него какой-нибудь текст.

Откройте файл todolist.js и добавьте к методу `TodoList.invokeSavePicker` следующий код, предназначенный для работы с компонентом выбора сохраняемого файла:

```
savePicker.pickSaveFileAsync().then(function (file) {
   if (file) {
     // Файл УЖЕ существует, компонент СОЗДАЕТ файл длиной 0 байтов
     Windows.Storage.FileIO
     .writeTextAsync(file, "Some data")
     .done(function () {
         TodoList.alert("Data successfully saved");
       },
       function (error) {
         TodoList.alert("Unable to save data. Sorry about that!");
       }
     );
   }
});
```

По правде говоря, синтаксис объектов ввода-вывода в Windows 8 довольно сложный, но это не страшно. Основным объектом для манипуляций при вводе-выводе является

`Windows.Storage.FileIO`. В этом объекте есть метод под названием `writeTextAsync`, который получает файловый объект и текст, а затем просто записывает текст в файл. В показанном примере в выбранном файле сохраняется текст «`Some data`».

Большинство системных операций в приложениях Магазина Windows выполняется в асинхронном режиме. Это означает, что инструкции, следующие за асинхронным вызовом, выполняются немедленно, не ожидая завершения других инструкций. Однако наряду с тем, что асинхронные вызовы гарантируют максимально быструю реакцию пользовательского интерфейса приложения, для нижележащего кода они усложняют чтение и запись.

В частности, чтобы выразить последовательную семантику, когда два и более действия выполняются последовательно, а каждое действие дожидается завершения предыдущего, необходимо обратиться к такому последовательному синтаксису:

```
Windows.Storage.FileIO
                .writeTextAsync( ... )
               .done( ok, error )
```

Код читается так: вы записываете какой-то контент в заданный файл, после чего выполнение программы идет разными путями в зависимости от того, успешно или нет была выполнена эта операция. Подобная схема в приложениях Магазина Windows используется довольно часто.

На рис. 10.2 показано текущее состояние папки после создания файла.

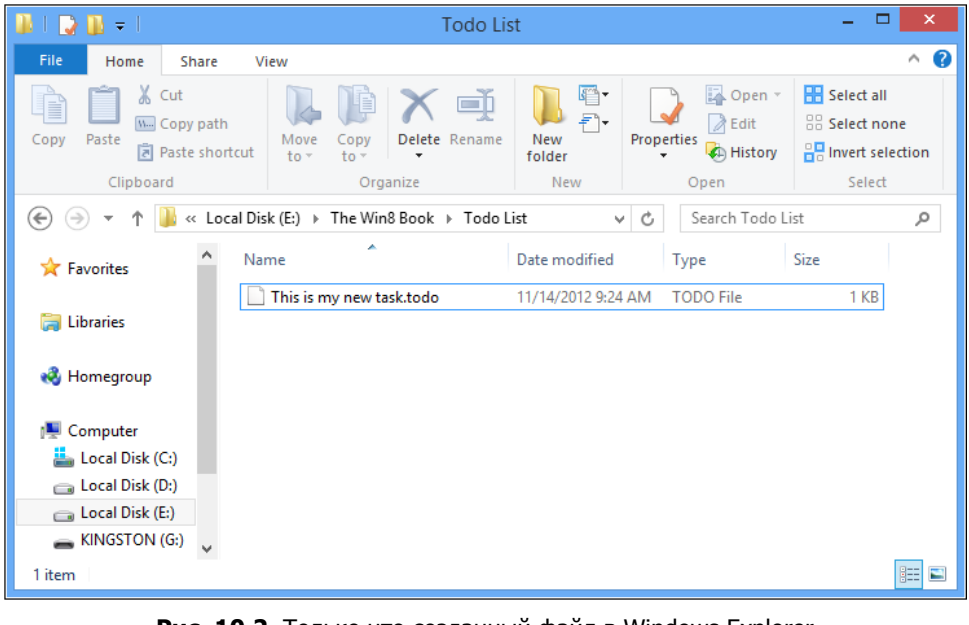

Рис. 10.2. Только что созданный файл в Windows Explorer

Создание собственных файлов

Если в следующий раз, работая с приложением TodoList, вы пройдете через процедуру добавления задания и выберете тот же самый файл, компонент выбора файла выведет сообщение, показанное на рис. 10.3.

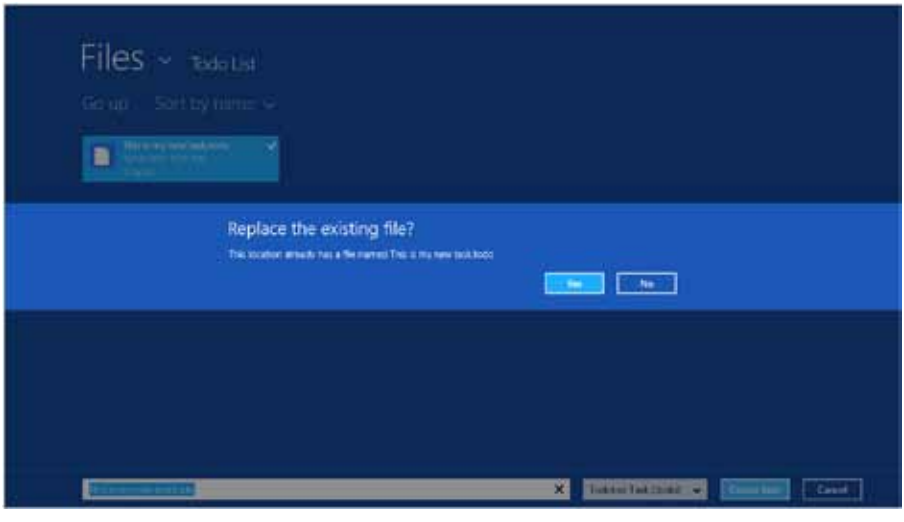

Рис. 10.3. Попытка перезаписи существующего файла

Это означает, что компонент выбора файла пытается создать файл, но обнаруживает, что файл с таким именем уже существует. После чего он спрашивает, как ему лучше поступить. Давайте посмотрим на прикладной программный интерфейс, который требуется для создания нового файла программным путем. В следующем примере предоставляется альтернативный путь выбора файла. Вместо использования компонента выбора сохраняемого файла вы выбираете папку, а затем программно создаете файл:

```
TodoList.invokeFolderPicker = function (task) {
    var folderPicker = new Windows.Storage.Pickers.FolderPicker();
    folderPicker.fileTypeFilter.replaceAll(["*"]);
    folderPicker.SuggestedStartLocation =
      Windows.Storage.Pickers.PickerLocationId.desktop;

    // Вызов компонента выбора папки
    var fileOptions = Windows.Storage.CreationCollisionOption;
    var io = Windows.Storage.FileIO;
    folderPicker
        .pickSingleFolderAsync()
        .done(
          function (folder) {
            if (folder) {
              folder.createFileAsync(
```

продолжение ⇩

```
                "sample.todo", fileOptions.replaceExisting)
              .done(function (file) {
                io.writeTextAsync(file, "Some more data.")
              .then(function () { io.appendTextAsync(file, " By me."); })
              .done(function () { TodoList.alert("All done!")})
            },
            function (error) {
              TodoList.alert("Unable to create file. Sorry about that!");
            });
        }
    });
}
```

Имея в своем распоряжении объект папки, можно вызвать метод `createFileAsync`, которому передается имя файла и ряд параметров. В частности, перечисление `Windows.Storage.CreationCollisionOption` включает в себя значения для молчаливой замены файла, если файл с таким именем уже существует (`replaceExisting`), или для отказа от операции при обнаружении конфликта имен (`failIfExists`).

Код для записи текста помещается в метод `done`, который запускается после создания файла. Кроме того, метод `done` может применяться для запуска любого последовательного кода, например, для вывода на экран пользователя сообщения о завершении операции.

ПРИМЕЧАНИЕ

Метод then выполняется либо в случае успешного завершения предыдущей функции, либо в случае ошибки. Метод done выполняется аналогичным образом, за исключением того, что он гарантирует выброс наружу любой ошибки, которая не обрабатывается внутри функции.

Добавление текста и простая запись текста

При использовании метода `writeTextAsync` Windows 8 просто открывает файл, записывает в него контент, а затем закрывает этот файл. Если вызвать метод `writeTextAsync` дважды, контент перезаписывается, поскольку второй вызов полностью уничтожает предыдущий контент.

Если нужно просто добавить текст к существующему файлу, следует воспользоваться методом `appendTextAsync`. Этот метод имеет точно такой же синтаксис, как и метод `writeTextAsync`. В предыдущем фрагменте кода показан пример поочередного вызова этих двух методов.

Однако в силу асинхронности прикладного программного интерфейса нужно стремиться свести количество последовательно выполняемых задач к минимуму, чтобы в коде не был достигнут такой уровень вложенности функций, которым уже невозможно будет управлять. Что касается записи в файл, то всегда нужно стараться записать любой контент в файл за один шаг.

ПРИМЕЧАНИЕ

Если внимательно изучить методы файлового объекта в Windows 8, то там найдутся методы для переименования, копирования, удаления и перемещения файлов. Но вы не найдете там методов для чтения или записи контента в файл, предлагаемый самим файловым объектом. Для чтения и записи файлов всегда нужно использовать методы объекта Windows.Storage.FileIO.

Удаление файлов

Другой весьма распространенной операцией, выполняемой приложением, является удаление ранее созданных им файлов, или, в более широком смысле, удаление любых файлов, которые ему по той или иной причине необходимо удалить. Совсем недавно упоминалось, что удаление файлов является операцией, требующей для своего выполнения файлового объекта. Рассмотрим пример, в котором предполагается, что доступный файловый объект существует:

```
file.deleteAsync()
   .done(function () { TodoList.alert("File deleted" },
        function () { TodoList.alert("Unable to delete the file" });
```

Выбор формата сериализации

Теперь, после знакомства с основами работы с файлами в Windows 8, давайте вернемся к приложению TodoList. Вы уже ввели в код механизм сохранения файла, но еще не сохранили ни одного задания. Фактически, ранее рассмотренный код предоставил возможность создать файл с заданным именем, и все свелось к сохранению в нем простого текста. Поэтому следующим шагом станет выбор формата для сохранения в файле реальных данных. Процесс оптимизации реальных данных при их записи в файл часто называют *сериализацией* (serialization).

Добавление компонента сериализации данных

Для того чтобы заставит метод `TodoList.invokeSavePicker` сохранить текущий объект `Task`, в него нужно внести некоторые изменения. Сначала нужно изменить строку, в которой вызывается метод `writeTextAsync`, чтобы вместо записи постоянного текста, взятого для примера, теперь записывалась строка, возвращаемая новой функцией-оболочкой:

```
Windows.Storage.FileIO
  .writeTextAsync(file, TodoList.serializeTask(task))
```

Кроме того, нужно создать заготовку функции `TodoList.serializeTask` и добавить ее в файл todolist.js:

```
// Функция сериализации задания
 TodoList.serializeTask = function (task) {
   // Просто сохраняем описание задания
   return task.description;
}
```

В данном случае в файле сохраняется только описание задания, а вся остальная информация теряется. А как сохранить весь набор данных объекта `Task`?

Проще всего ответить на этот вопрос, сказав, что формат сериализации зависит от вас. Нужно лишь знать, что функция `TodoList.serializeTask` возвращает строку, которая в неизменном виде сохраняется в файле. С точки зрения функции подойдет любая строка.

Объект `Task` состоит из нескольких информационных частей: описания, даты выполнения, приоритета и т. д. Вам может потребоваться сохранить всю эту информацию. Для объединения отдельных информационных частей можно выбрать в качестве разделителя какой-нибудь конкретный символ (например, символ вертикальной черты). Рассмотрим пример:

```
TodoList.serializeTask = function (task) {
   return task.description + "|" +
          task.dueDate + "|" +
          task.priority + "|" +
          task.status + "|" +
          task.percCompleted;
}
```

Получившаяся строка может иметь следующий вид:

```
This is my new task|Sat Dec 1 12:00:00 UTC+0100 2012|3|Not Started|0
```

Этот формат не вызывает никаких возражений, если не считать того, что для него требуется специальный фрагмент кода, выполняющий чтение и синтаксический разбор строки с целью получения исходного объекта `Task`. Процесс превращения строки текста в объект называется *десериализацией* (deserialization).

В общем, проблема заключается в выборе нестандартного формата сериализации, что делает вас ответственным за написание процедур сериализации и десериализации. Кроме того, каждый формат требует собственной специальной пары таких процедур.

JavaScript предлагает более подходящий подход — специальную нотацию, известную как формат JSON (JavaScript Object Notation — нотация JavaScript-объектов).

Формат JSON

Текстовый формат JSON прост в обращении и понятен как людям, так и программам. По сути, в формате JSON определяется набор соглашений, на основе которого

данные любого объекта могут быть сериализованы в стандартный формат. JSON-строка состоит из коллекции пар имя-значение, с помощью которых идентифицируются имена свойств и их значения. Также существует специальное соглашение для визуализации массивов. В следующем тексте показана JSON-версия объекта `Task`, заполненного значениями, предлагаемыми по умолчанию:

```
{
   "description":"This is my new task",
   "dueDate":"2012-12-01T11:00:00.000Z",
   "priority":"3",
   "status":"Not Started",
   "percCompleted":"0"
}
```

Для нас здесь важно то, что JavaScript и Windows 8 предоставляют встроенные средства создания JSON-строки из JavaScript-объекта, а также для получения JavaScript-объекта из JSON-строки. Кроме того, универсальность формата JSON позволяет применять его практически для любого объекта. Это означает, что в вашем коде независимо от количества и структуры сохраняемых объектов нужна только одна пара процедур сериализации-десериализации.

Сериализация объекта Task в формат JSON

Для использования формата JSON в приложении TodoList достаточно включить в функцию `TodoList.serializeTask` следующий код:

```
TodoList.serializeTask = function (task) {
   return JSON.stringify(task);
}
```

Функция `JSON.stringify` является «родной» JavaScript-функцией, превращающей любой JavaScript-объект в строку, совместимую с форматом JSON. На рис. 10.4 показан JSON-контент объекта `Task`, считанный простым текстовым редактором, таким как Блокнот.

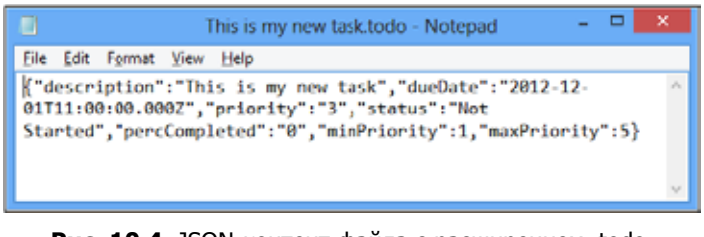

Рис. 10.4. JSON-контент файла с расширением .todo, созданного приложением TodoList

Теперь предстоит решить еще одну задачу: научить приложение TodoList загружать и редактировать существующие задания, хранящиеся в формате JSON.

Создание объектов Task из файлов

Один из способов обратного чтения контента сохраненных объектов `Task` предполагает внесение двух изменений в пользовательский интерфейс приложения TodoList. Нужно добавить кнопки для создания нового пустого задания и открытия существующего задания из указанного файла.

Внесение изменений в пользовательский интерфейс

Откройте файл default.html и добавьте к элементу `div`, содержащему две кнопки для добавления и обмена данными задания, следующую разметку:

```
<button id="buttonNewTask">New</button>
<button id="buttonOpenTask">Open</button>
```

Затем найдите в файле todolist.js метод `TodoList.init` и добавьте в него следующий код, регистрирующий обработчики для события `click` этих двух новых кнопок:

```
document.getElementById("buttonNewTask").addEventListener(
    "click", TodoList.newTaskClick);
 document.getElementById("buttonOpenTask").addEventListener(
  "click", TodoList.openTaskClick);
```

А вот как выглядят тела этих обработчиков, которые нужно включить в тот же JavaScript-файл:

```
TodoList.newTaskClick = function () {
    TodoList.displayTask(new Task());
 }

 TodoList.openTaskClick = function () {
    TodoList.pickFileAndOpenTask();
}
```

Обработчик события `click` для кнопки New (Новое задание) предельно прост: он создает новый пустой объект `Task` и выводит его на экран с помощью вспомогательного метода `displayTask`. Обработчик события `click` для кнопки Open (Открыть) немного сложнее, поскольку он связан с компонентом выбора файла, с помощью которого определяется местоположение и происходит выбор открываемого файла.

Определение местонахождения открываемого файла

Код метода `TodoList.pickFileAndOpenTask` похож на код, который мы ранее использовали для компонента выбора сохраняемого файла. На этот раз мы настраиваем объект выбора файла на файлы с расширением .todo. Эта функция также должна быть добавлена в файл todolist.js:

```
TodoList.pickFileAndOpenTask = function () {
    var currentState = Windows.UI.ViewManagement.ApplicationView.value;
    if (currentState ===
      Windows.UI.ViewManagement.ApplicationViewState.snapped &&
```

```
      !Windows.UI.ViewManagement.ApplicationView.tryUnsnap()) {
      // Молчаливый отказ в случае невозможности открепления
      return;
    }

    TodoList.invokeOpenPicker();
}
```

Сначала код проверяет, что приложение не закреплено, а затем вызывает еще одну вспомогательную функцию, работающую с компонентом выбора открываемого файла. Следует заметить, что выбор файлов можно осуществлять только в режиме заполнения или в полноэкранном режиме, кроме того, компонент выбора файлов не может вызваться из закрепленного приложения и из всплывающих меню, включая меню настройки. Код вспомогательной функции `TodoList.invokeOpenPicker` выглядит следующим образом:

```
TodoList.invokeOpenPicker = function () {
    var openPicker = new Windows.Storage.Pickers.FileOpenPicker();
    openPicker.suggestedStartLocation =
      Windows.Storage.Pickers.PickerLocationId.computerFolder;
    openPicker.fileTypeFilter.replaceAll([".todo"]);
    openPicker.pickSingleFileAsync().then(function (file) {
      if (file) {
        // Что-то делаем с выбранным файлом с расширением .todo
      }
    });
}
```

После выбора пользователем файла с расширением .todo код должен считать весь контент и попытаться создать из него объект `Task`. Как уже упоминалось файл с расширением .todo содержит текст в формате JSON, полученный в результате предшествующей сериализации объекта `Task`.

Чтение контента файла

Для чтения контента выбранного файла служит метод `readTextAsync` объекта `Windows.Storage.FileIO`. Этот метод возвращает весь контент файла в виде строки текста. Кроме того, объект `Windows.Storage.FileIO` предоставляет еще два метода. Один из них, `readLinesAsync`, считывает весь контент и возвращает его в виде массива строк. Какой метод использовать, `readLinesAsync` или `readTextAsync`, зависит от того, что вы планируете делать со считанным текстом. Для десериализации JSON-контента нам подходит последний метод, поскольку мы не собираемся разбивать текст на строки.

Еще одним методом, доступным в объекте `Windows.Storage.FileIO`, является `readBufferAsync`. Этот метод возвращает буферный объект, то есть массив байтов. Он может пригодиться при работе с двоичными файлами, например с изображениями. Для приложений, работающих с текстом, этот метод не нужен.

Чтобы получить JSON-строку, сохраненную в выбранном файле, к методу `TodoList.invokeOpenPicker` нужно добавить следующий код:

```
openPicker.pickSingleFileAsync().then(function (file) {
    if (file) {
      var io = Windows.Storage.FileIO;
      io.readTextAsync(file)
        .done(function (json) {
        // Сюда помещается код JSON-десериализации
      }
    }
});
```

Заключительным шагом станет десериализация JSON-строки во вновь созданный объект `Task`.

Десериализация объектов Task

Чтобы сериализировать объект `Task` в строку, используется метод `JSON.stringify`. Существует также и обратный метод для восстановления JavaScript-объекта из JSON-строки. Этот метод называется `JSON.parse`. К методу `TodoList.invokeOpenPicker` нужно добавить следующий код:

```
// Создание объекта Task
 var task = JSON.parse(json);

 // Вывод объекта Task на экран
TodoList.displayTask(task);
```

К сожалению, при выполнении этого кода возникает исключение, связанное с невозможностью создания даты для объекта `Task`. Как бы странно это ни звучало, но то, что было сериализировано методом `JSON.stringify`, нельзя правильно десериализировать его партнером — методом `JSON.parse`.

Точнее говоря, этот конфликт проявляется только в том случае, если сериализуемый объект имеет свойства типа `Date`. Если даты не используются, то все проходит без сбоев. Проблема кроется в спецификации JSON, в которую официально не включен тип данных `Date`. И главная проблема заключается не в том, что метод `JSON.parse` не может обработать строку даты, а в том, что дата при десериализации превращается в обычную строку. Стало быть, проблемы возникают не при десериализации, а при работе с десериализованными данными (если такая работа выполняется). Иными словами, исключения могут возникать в тех местах вашего кода, которые не имеют прямого отношения к десериализации.

Как же справиться с этой проблемой?

Все можно исправить быстро и легко, но применять это исправление придется при каждом вызове:

```
// jsonDate является строкой, получаемой для даты из JSON.parse.
 // Для получения объекта даты эта строка передается
 // конструктору объекта Date
var date = new Date(jsonDate);
```

Это исправление следует применять, когда приложение выполняет десериализацию из формата JSON и когда используется дата. Окончательная версия кода для метода `TodoList.invokeOpenPicker`:

```
openPicker.pickSingleFileAsync().then(function (file) {
    if (file) {
      var io = Windows.Storage.FileIO;
      io.readTextAsync(file)
        .done(function (json) {
          var task = TodoList.deserializeTask(json);
          TodoList.displayTask(task);
        },
        function () { TodoList.alert("Unable to read the file") });
    }
});
```

Сначала выполняется десериализация JSON-строки в объект `Task`, а затем исправляются свойства типа `Date`. Чтобы сделать код понятнее, лучше в файле todolist.js поместить код десериализации в специальный метод:

```
TodoList.deserializeTask = function (json) {
    var task = JSON.parse(json);
    task.dueDate = new Date(task.dueDate);
    return task;
}
```

Рисунок 10.5 иллюстрирует процесс выбора задания, а на рис. 10.6 выбранное задание показано в пользовательском интерфейсе приложения.

Рис. 10.5. Выбор файла задания

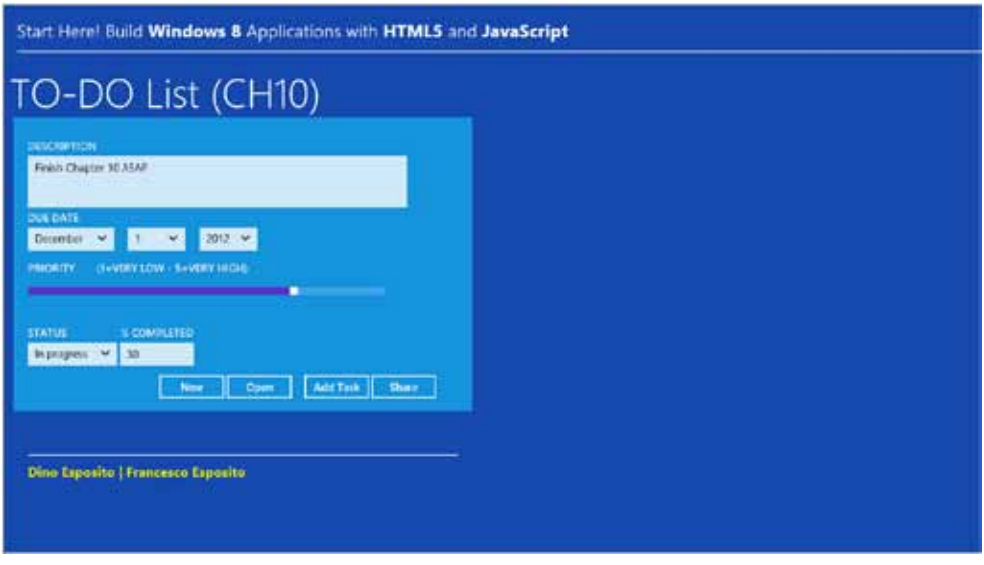

Рис. 10.6. Выбранное задание в пользовательском интерфейсе приложения TodoList

Использование собственного хранилища приложения

До сих пор создание и использование файлов было связано с непосредственным доступом к локальному диску машины. Как вы уже, наверное, заметили, приложению Магазина Windows не всегда разрешается полный доступ к диску — свободный доступ по чтению и записи возможен лишь к части диска. Далее в этой главе мы будете работать с другой формой хранилища, занимающего ту часть диска, которая принадлежит конкретному приложению и абсолютно невидима другим приложениям.

Варианты хранилищ в Windows 8

Что касается хранилищ, у приложений Магазина Windows есть три варианта. Ранее в главе 9 нам уже приходилось использовать хранилище локальных параметров, а в этой главе мы сохраняли файлы на локальном диске. Это два из трех вариантов хранилищ для приложений Магазина Windows. Третий вариант представляет собой часть локального диска, зарезервированную для приложения. Давайте проведем краткое сравнение этих трех вариантов, чтобы понять, для каких случаев лучше подходит каждый из них.

Сохранение данных в хранилище локальных параметров

Хранилище локальных параметров является контейнером данных приложения, имеющим форму словаря. Доступ к нему осуществляется программно:

```
var applicationData = Windows.Storage.ApplicationData.current;
var localSettings = applicationData.localSettings;
```

Чтение и запись данных осуществляются с использованием классических словарных подходов. Каждая запись имеет уникальный идентификатор в виде ключа, с помощью которого осуществляется ссылка на значение. Ключ представляет собой строку длиной до 255 символов, а значение может быть данными любого допустимого в Windows 8 типа (включая массивы и коллекции нестандартных типов) длиной не более 8 Кбайт. Словарь `localSettings` уже использовался в главе 9 для хранения применяемых по умолчанию параметров приложения TodoList. Запись и чтение данных словаря производятся следующим образом:

```
// Запись в хранилище
 localSettings.values["sampleKey"] = "Some data";

 // Чтение из хранилища
var value = localSettings.values["sampleKey"];
```

Словарь `localSettings` предлагает довольно богатый программный интерфейс с методами для удаления записей, для проверки существования записи, а также для выполнения запросов. Дополнительные сведения можно получить в документации MSDN для класса `ApplicationDataContainerSettings` по URL-адресу http://msdn.microsoft.com/en-us/library/windows/apps/windows.storage.applicationdatacontainersettings.

В хранилище локальных параметров могут храниться не только пользовательские предпочтения, но и оперативные данные приложения. Но из-за словарного формата и ограничений в размере отдельных записей это возможно только для небольших блоков данных, которые могут извлекаться и храниться в виде пар ключ-значение. Такое хранилище может быть полезным, если нужные данные можно извлекать путем непосредственного вызова. К тому же при использовании этого прикладного программного интерфейса не придется сталкиваться со сложностями асинхронного программирования и основными операциями ввода-вывода, такими как создание, открытие и поиск файлов.

ВНИМАНИЕ

Если вместо словаря localSettings использовать словарь roamingSettings, то все сохраняемые вами данные будут автоматически синхронизированы Windows 8 для всех устройств и компьютеров, выполняющих приложение под управлением той же учетной записи Windows. Это означает, что вы можете задать свои предпочтения, к примеру, на устройстве Microsoft Surface, а применять их при работе с тем же приложением на персональном компьютере под управлением Windows 8. Прикладные программные интерфейсы чтения и записи данных у словарей roamingSettings и localSettings идентичны. Единственным препятствием для использования словаря roamingSettings в качестве применяемого по умолчанию хранилища является ситуация, когда сохраняемые данные на другом устройстве бесполезны. Но обычно это не относится к параметрам, представляющим собой пользовательские предпочтения относительно конфигурации того или иного приложения.

Сохранение на локальном диске

Вы уже поняли, что приложение Магазина Windows может получать частичный доступ к локальному диску. Получать программный доступ к папке Documents или Pictures можно, считывая и создавая файлы в этих папках. Чтобы иметь доступ к любому файлу практически в любой папке, можно задействовать компонент выбора файла. Однако вы не сможете программно заставить код обращаться к корневой папке диска и создавать там файлы и папки. Для полного доступу к локальному диску необходимы компоненты выбора файлов.

Важно отметить, что наряду с разрешением программного доступа к таким известным папкам, как Documents и Pictures, для такого доступа все равно требуется одобрение пользователя. Точнее, приложение Магазина Windows, которому требуется работать с известной папкой, должно заранее объявить об этом, тогда операционная система будет уведомлять об этом пользователя, устанавливающего приложение. Для этого нужно открыть в Microsoft Visual Studio принадлежащий проекту файл манифеста и перейти на вкладку Capabilities (Возможности), как показано на рис. 10.7.

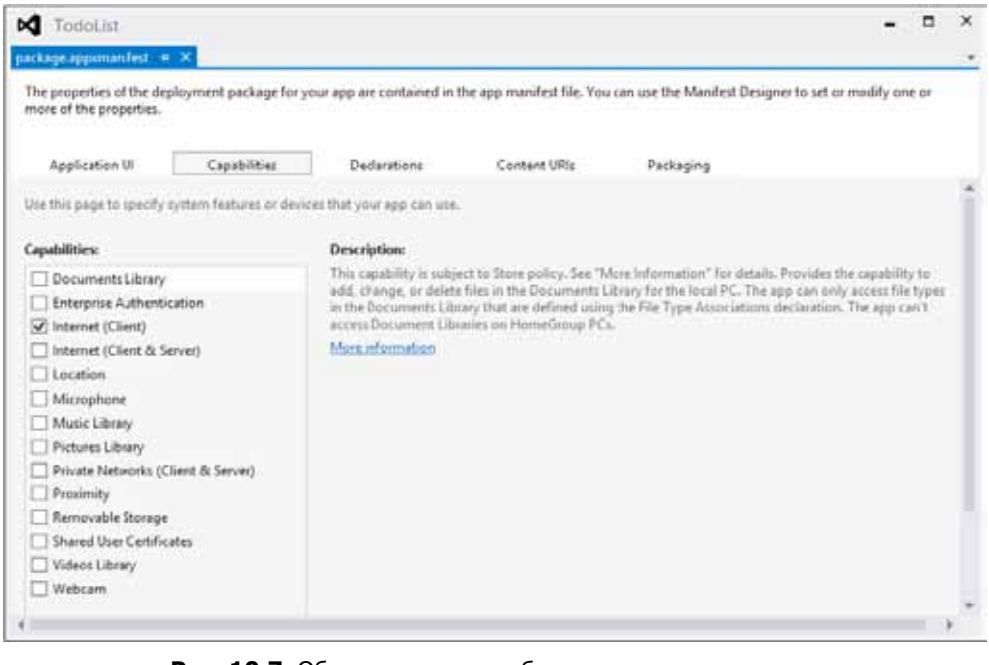

Рис. 10.7. Объявление о затребованных для приложения Магазина Windows возможностях

За некоторыми исключениями, касающимися библиотечных папок (Documents, Pictures, Music), использование локального диска из приложения Магазина Windows

при работе с локальными файлами лимитировано компонентами выбора файлов. Но при получении ссылки на файл или папку с ними можно вполне свободно работать. Вариант с локальным диском подходит для тех приложений, которые предназначены для создания и использования данных, остающихся на диске для применения другими приложениями. Если данные, с которыми работает ваше приложение, за его пределами никакого смысла не имеют, то непосредственное обращение к локальному диску будет, наверное, не самым удачным вариантом.

ПРИМЕЧАНИЕ

При удалении приложения папки на локальном диске не удаляются. Для приложения этот аспект может быть аргументом как за, так и против, в зависимости от предполагаемого назначения приложения.

ВНИМАНИЕ

Как показано на рис. 10.7, приложение Магазина Windows должно также объявить о своей способности манипулировать файлами на таких сменных запоминающих устройствах, как внешний жесткий диск или USB-накопитель. Кроме того, приложение, предназначенное для поддержки съемных накопителей, должно объявить на вкладке Declarations (Объявления) манифеста типы файлов, с которыми оно собирается работать. Это делается с целью не допустить доступ приложения к любым другим файлам, которые могут быть у пользователя на внешнем устройстве.

Сохранение данных в изолированном хранилище

Иногда приложение и его данные формируют монолит, который пользователи хотели бы задействовать или целиком удалить. Такая схема является своего рода новинкой для разработчиков, привыкших создавать классические приложения для настольных систем, но она должна быть знакома тем, кто имеет опыт разработки программ для мобильных устройств и приложений Microsoft Silverlight.

Приложения Магазина Windows могут также иметь собственное закрытое пространство для создания файлов и папок. И хотя это пространство физически находится на локальном диске, к его содержимому невозможно получить доступ за пределами приложения, причем ни программный, ни через компонент выбора файла. Все файлы и папки, созданные в закрытом пространстве приложения, утрачиваются, когда пользователь удаляет приложение. Закрытое пространство приложения часто называют *изолированным хранилищем* (isolated storage).

Существует три типа изолированных хранилищ: локальные, роуминговые и временные (табл. 10.1).

Таблица 10.1. Типы изолированных хранилищ

Хранилище	Описание
Локальное	Файлы и папки, созданные в локальном хранилище, сохраняются только на локальной машине и находятся в хранилище, пока пользователь не удалит приложение. Всю ответственность за обновление или удаление файлов несет приложение. Эти файлы недоступны другим приложениям и компонентам выбора файлов
Роуминговое	Файлы и папки, созданные в роуминговом хранилище, подчиняются тем же правилам, которые действуют для локального хранилища, за исключением того, что они синхронизируются для всех машин с Windows 8, выполняющим ваше приложение под управлением той же самой учетной записи Windows
Временное	Файлы и папки, созданные во временном хранилище, подчиняются тем же правилам, которые действуют для локального хранилища, за исключением того, что они периодически могут удаляться системой Windows

Прикладной программный интерфейс для работы с файлами (то есть для их создания, удаления и чтения их контента) на локальном диске или на изолированном хранилище один и тот же. Все изменения касаются корневого объекта, к которому применяются ваши действия по вводу-выводу данных.

ПРИМЕЧАНИЕ

Если ни один из перечисленных здесь вариантов хранения данных вам не подходит, придется, наверное, рассмотреть вопрос использования баз данных. База данных позволяет сохранять данные в таблицах, являющихся коллекциями взаимосвязанных данных, упорядоченных по столбцам. Например, наш знакомый объект Task может быть легко представлен в виде строки в таблице базы данных. Положительной стороной баз данных, главным образом, является возможность создавать запросы, а в нашем случае база данных могла бы предложить беспрецедентные возможности поиска заданий по их приоритету, сроку выполнения или, может быть, отметке о выполнении. Если эти стороны имеют для вас ключевое значение, было бы целесообразно включить в ваше приложение Магазина Windows базу данных. Но тогда возникает вопрос, какую базу данных можно запустить на устройстве под управлением Windows 8? Наверное, лучшим выбором будет SQLite. Рабочий модуль SQLite для приложений Магазина Windows можно установить непосредственно из меню Visual Studio, выбрав команду Tools (Инструментарий) ▸ Extensions and Updates (Расширения и обновления). Далее нужно будет выбрать вариант Online (В подключенном режиме) и перейти на вкладку Visual Studio Gallery (Галерея Visual Studio), а затем запросить SQLite. Детали процесса установки и краткое знакомство с использованием SQLite в Windows 8 можно найти по адресу http://bit.ly/MuzL1e.

Создание заданий в изолированном хранилище

В завершение данной главы мы создадим новую версию приложения TodoList, сохраняющего задания в изолированном хранилище, причем конфигурация должна

обеспечивать роуминг заданий между несколькими устройствами, работающими под управлением Windows 8. Это означает, что ваши пользователи смогут создать задание на настольном компьютере с Windows 8, а получить его в приложении, запущенном с устройства Microsoft Surface.

Переделка пользовательского интерфейса приложения TodoList

Чтобы сохранить в неизменном состоянии текущую версию TodoList, в которой используются компоненты выбора файлов и создаются задания, которые можно сохранять в любом месте на диске, давайте создадим новую копию приложения. Для этого можно скопировать всю папку и дать ей другое имя, например Todolist-Local. Чтобы не возникло путаницы, придется также изменить панель заголовка в файле default.html, заменив элемент `h1` следующей разметкой:

```
<h1>TO-DO List (CH10-Local)</h1>
```

Основной особенностью этой новой версии является полный отказ от компонентов выбора файлов. Соответственно, кнопка Open (Открыть) больше не нужна, но в то же время понадобится представление, в котором перечисляются все текущие задания, чтобы их можно было выбрать для последующего редактирования. В файл default.html сразу же после элемента `h1` нужно добавить такую разметку:

```
<div id="list-of-tasks">
   <button id="buttonNewTask">New</button>
   <div id="task-listview"
     data-win-options ="{layout: {type: WinJS.UI.Listayout}}"
     data-win-control="WinJS.UI.ListView">
   </div>
 </div>
 <div id="task-listitem" data-win-control="WinJS.Binding.Template">
   <div class="listitem"><span data-win-bind="innerText: description"></span></div>
</div>
```

Кроме того, нужно последнему элементу `div` страницы присвоить класс `form-container` и удалить элементы `button` с метками `New` и `Open`. Следует уточнить, что кнопка New (Создать) была перемещена в только что добавленный элемент `div`.

Кроме того, может понадобиться поместить видимые элементы управления, используемые для создания задания, в новый элемент `div`, идентифицированный как `editor-container`:

```
<div id="task-editor" class="form-container">
   <div id="editor-container">
     <div id="buttonCancel-container"><button id="buttonCancel">Cancel</button></
  div>
     <!-- Сюда помещается существующий редактор задания -->
   </div>
</div>
```

Этот дополнительный контейнер служит для того, чтобы дать возможность скрывать и показывать редактор, сохраняя при этом графические параметры внешнего

контейнера. В верхней части дополнительного контейнера `div` помещается еще один контейнер `div` для кнопки Cancel (Отмена).

Чтобы завершить переделку пользовательского интерфейса, нужно внести небольшие дополнения в файл default.css, чтобы придать новым элементам некоторые графические особенности, отличающиеся от предлагаемых по умолчанию:

```
#list-of-tasks {
   float: left;
   width: 300px;
   height: 480px;
   color: #eee;
   padding: 5px;
   background-color: #1593dc;
   margin-top: 20px;
   margin-left: 20px;
 }
 #list-of-tasks button {
   margin: 5px;
 }
 #task-listview {
   background-color: #daf5f7;
   height: 440px;
 }
 .listitem {
   color: #eee;
   padding: 4px;
   background-color: #00f;
   width: 280px;
 }
 #task-editor {
   float: left;
   margin-top: 48px;
 }
 #buttonCancel-container {
   text-align: right;
 }
 #editor-container {
   display: none;
}
```

Только что созданный новый пользовательский интерфейс показан на рис. 10.8.

Но это еще не все! Чтобы действительно получить такой результат, нужно в файл todolist.js добавить еще один сценарий.

Замысел заключается в том, что пользователь выбирает задание в списке в левой части окна и редактирует его в панели, расположенной в правой части окна. Если пользователь хочет создать новое задание, он щелкает на кнопке New (Создать), расположенной в верхней части списка. Следовательно, больше не нужно при запуске приложения создавать и выводить на экран пустое задание. На данный момент все изменения касаются только файла TodoList.init.

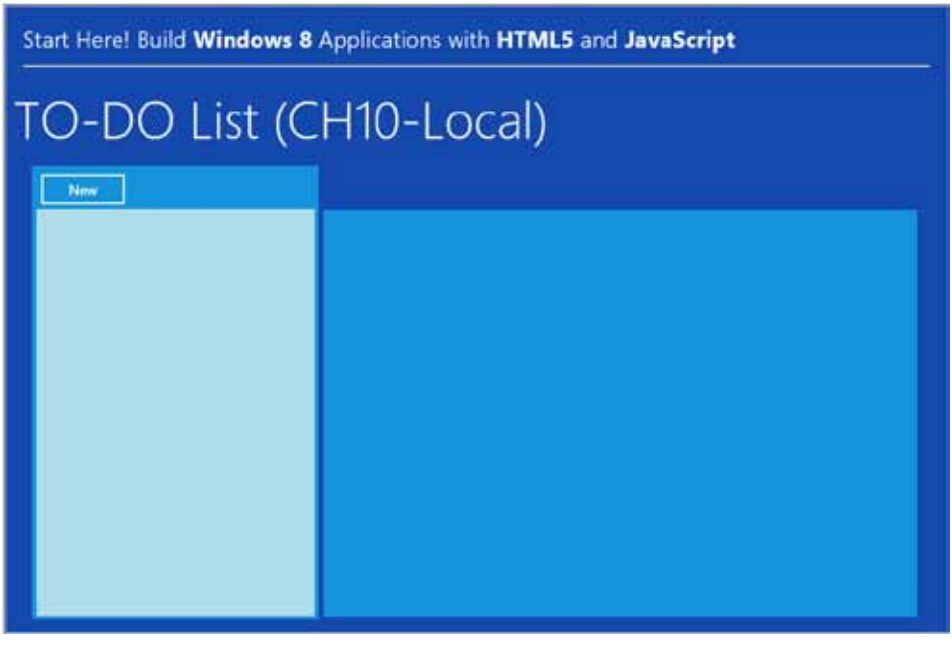

Рис. 10.8. Новый пользовательский интерфейс приложения TodoList

Сначала нужно убедиться в том, что в начале метода есть показанная далее строка. Вполне логично предположить, что эта строка осталась на своем месте, потому что все изменения по сравнению со старой версией приложения касаются перемещения кнопки New (Создать) в другое место:

```
document.getElementById("buttonNewTask").addEventListener(
  "click", TodoList.newTaskClick);
```

В конце метода `TodoList.init` нужно также закомментировать (или просто удалить) следующую строку:

```
// TodoList.displayTask(new Task());
```

Метод `TodoList.displayTask` по-прежнему сохраняет ключевое значение для сценария и будет вызываться из других мест. В этот метод также требуется внести ряд изменений:

```
TodoList.displayTask = function (task) {
   TodoList.performInitialBinding(task);

   // Обеспечиваем видимость редактора
   var editor = document.getElementById("editor-container");
   editor.style.display = "block";
}
```

В частности, теперь методу `displayTask` нужно обеспечить видимость редактора задания, поскольку ранее внесенные в CSS-файл изменения просто делают редактор невидимым при запуске приложения.

Когда пользователь щелкает на кнопке New (Создать), он должен в ответ увидеть уже знакомый интерфейс приложения TodoList, показанный на рис. 10.9.

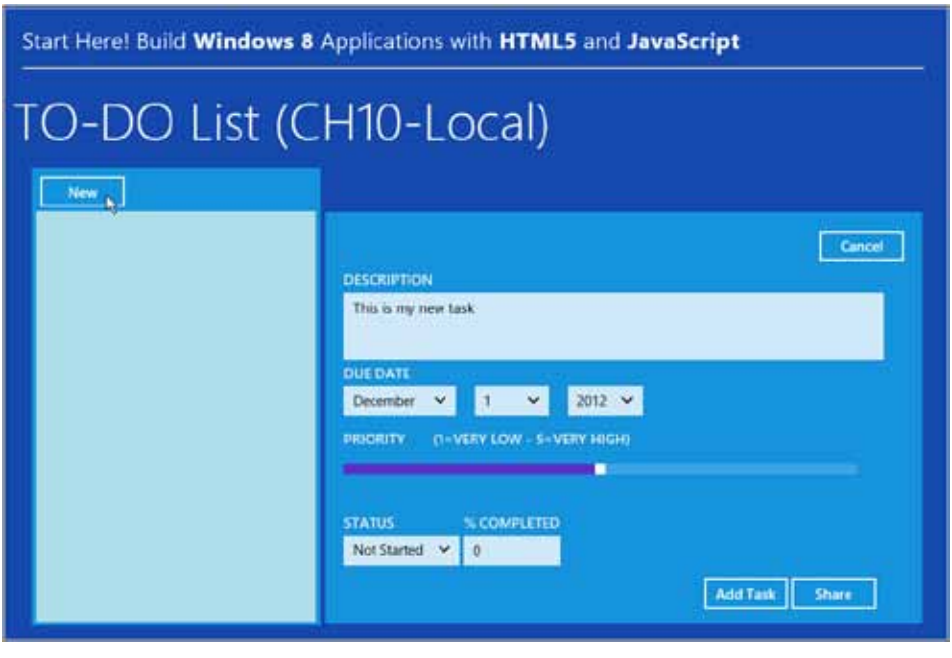

Рис. 10.9. Теперь, чтобы создать новое задание, пользователь щелкает на кнопке New

Кнопка Cancel (Отмена) предназначена для закрытия редактора, и для нее нужно зарегистрировать и создать обработчик события щелчка. Регистрация обработчика события щелчка на кнопке осуществляется обычным образом:

```
document.getElementById("buttonCancel").addEventListener(
  "click", TodoList.cancelTaskClick);
```

А вот как выглядит требуемая реализация обработчика:

```
TodoList.cancelTaskClick = function () {
    var editor = document.getElementById("editor-container");
    editor.style.display = "none";
}
```

С графическими изменениями покончено, и теперь все готово для решения более интересной задачи заполнения просматриваемого списка данными из любых файлов заданий, найденных в локальном хранилище.

Получение текущих заданий

В идеале при запуске приложения список нужно заполнить всеми доступными заданиями. Для этого в последней инструкции метода `TodoList.init` нужно добавить следующий вызов функции:

```
TodoList.populateTaskList();
```

Затем сделаем предварительную заготовку тела функции:

```
TodoList.populateTaskList = function () {
   var tasks = [
     { description: "Task #1" },
     { description: "Task #2" },
     { description: "Task #3" }
   ];

   var bindingList = new WinJS.Binding.List(tasks);
   var listview = document.getElementById("task-listview").winControl;
   listview.itemDataSource = bindingList.dataSource;
   listview.itemTemplate = document.getElementById("task-listitem");
}
```

Здесь берется статический массив объектов со свойством под именем `description` (описание), который привязывается к предварительно добавленной к странице в файле default.html переменной `ListView`. Свойство `description` является ключевым, поскольку на него ссылается шаблон списка. С поправками в файле default.html у вас должна быть следующая разметка, обеспечивающая визуализацию любых данных, привязанных к ListView:

```
<div id="task-listitem" data-win-control="WinJS.Binding.Template">
    <div class="listitem"><span data-win-bind="innerText: description"></span></
  div>
</div>
```

На рис. 10.10 показано текущее состояние приложения TodoList.

Следующий шаг заключается в изменении кода функции `populateTaskList`, чтобы она создавала список объектов `Task` из файлов, найденных в локальной папке приложения. Ссылка на локальную папку приложения осуществляется в следующем коде:

```
var localFolder = Windows.Storage.ApplicationData.current.localFolder;
```

Чтобы обеспечить доступность данных приложения для роуминга, достаточно сослаться на другую папку:

```
var localFolder = Windows.Storage.ApplicationData.current.roamingFolder;
```

Таким образом, создаваемый в дальнейшем код переделывать уже не придется, если данные нужно сохранять в локальной или в роуминговой папке машины.

Для получения списка файлов в папке служит метод `getFilesAsync`. Этот метод передает список файлов, найденных в папке, своей функции обратного вызова

then. Нужно учесть, что при наличии в папке других папок ни они, ни их контент в выходные данные метода getFilesAsync не включаются. Чтобы получить содержащиеся в папке другие папки и их контент, нужно вместо этого метода использовать метод getItemsAsync:

```
var localFolder = Windows.Storage.ApplicationData.current.roamingFolder;
 localFolder.getFilesAsync()
        .then(function (files) {
         var io = Windows.Storage.FileIO;
         files.forEach(function (file) {
           // Какие-нибудь действия с файлом
         });
       });
```

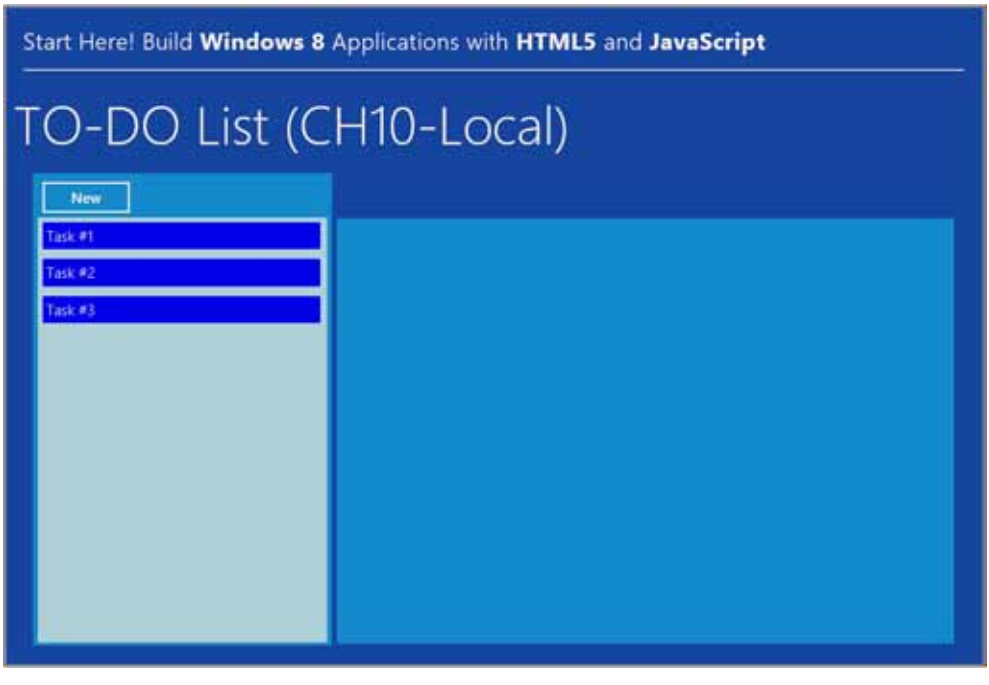

Рис. 10.10. Представление списка, заполненного имитаторами заданий

А что делать с извлеченным файлом? Предполагая, что файл содержит строку в формате JSON, нужно прочитать его контент, а затем десериализовать его в объект Task. Затем только что созданный объект нужно добавить в массив. И наконец, массив может быть превращен в связанный список и выведен на экран с помощью компонента ListView.

Полностью реализация метода TodoList.populateTaskList выглядит так:

```
var tasks = new Array();
 var localFolder = Windows.Storage.ApplicationData.current.roamingFolder;
```

```
 localFolder.getFilesAsync()
   .then(function (files) {
     var io = Windows.Storage.FileIO;
     files.forEach(function (file) {
       io.readTextAsync(file)
       .then(function (json) {
         var task = TodoList.deserializeTask(json);
         tasks.push(task);
       })
       .then(function () {
          var bindingList = new WinJS.Binding.List(tasks);
          var listview = document.getElementById("task-listview").winControl;
          listview.itemDataSource = bindingList.dataSource;
          listview.itemTemplate = document.getElementById("task-listitem");
       });
   });
})
```

Как можно заметить, из-за необходимости использовать в приложении асинхронные методы пришлось приложить дополнительные усилия, чтобы сделать код понятнее и правильно его отформатировать. Также следует иметь в виду, что любая операция над контентом папки должна выполняться внутри функции обратного вызова `then`.

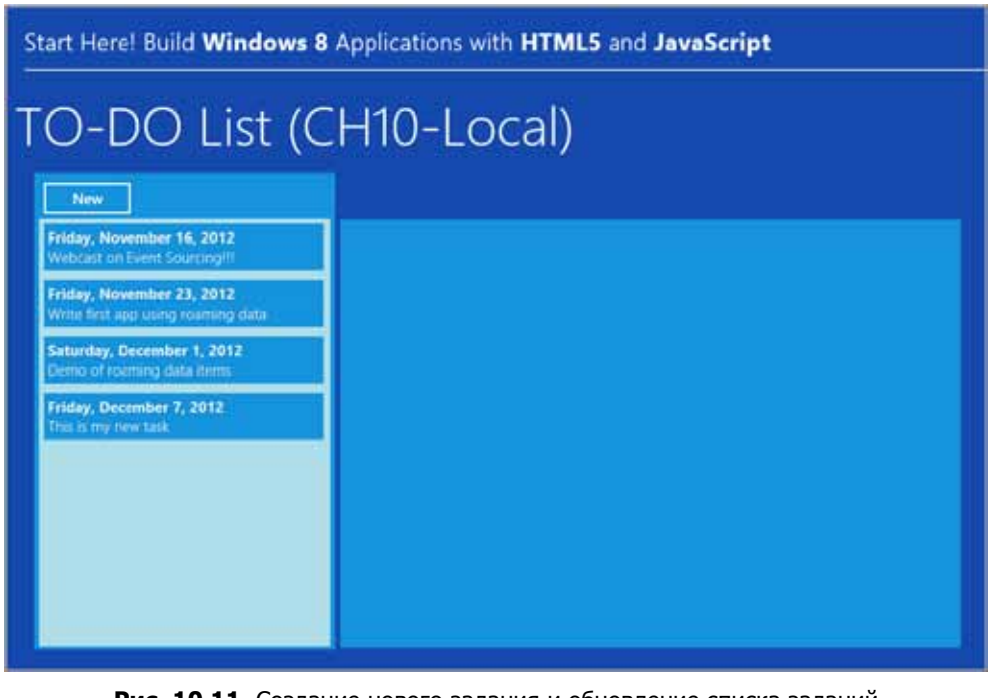

Рис. 10.11. Создание нового задания и обновление списка заданий

В качестве последнего штриха может понадобиться добавить сюда еще немного кода для сортировки заданий по дате. Это повлечет за собой добавление к объекту

bindingList двух строк кода. Код, который нужно добавить к заключительному блоку функции then, где осуществляется привязка, должен выглядеть так:

```
var bindingList = new WinJS.Binding.List(tasks);

 // А теперь сортировка списка перед привязкой к listview
 bindingList = bindingList.createSorted(function (first, second) {
   return first.dueDate > second.dueDate;
});
```

Кроме того, для позиций списка вам может понадобиться чуть более сложный шаблон. Вместо имени файла (или описания задания) вам может также понадобиться вывести на экран срок выполнения, чтобы сортировка имела для пользователя больший смысл. Вот модифицированный шаблон для позиций списка:

```
<div id="task-listitem" data-win-control="WinJS.Binding.Template">
   <div class="listitem">
     <span data-win-bind="innerText: dueDate TodoList.dateForDisplay"></span>
     <br />
     <span data-win-bind="innerText: description"></span>
   </div>
</div>
```

Вот теперь, как показано на рис 10.11, с выводом списка заданий все в порядке. Далее нужно привести в порядок код, который сохраняет задания в локальной или в роуминговой папке.

Сохранение заданий в роуминговой папке

Когда пользователь выполняет щелчок, чтобы добавить или обновить задание, текущая версия кода вызывает метод TodoList.pickFileAndSaveTask. Этот метод, в свою очередь, для поиска файла и сохранения в нем контента задания использует компонент поиска файлов. Можно сохранить то же имя файла и просто переписать его тело, придав ему следующий вид:

```
var task = TodoList.getTaskFromUI();

 var localFolder = Windows.Storage.ApplicationData.current.roamingFolder;
 var name = task.description;
 var io = Windows.Storage.FileIO;
 var fileOptions = Windows.Storage.CreationCollisionOption;
 localFolder.createFileAsync(name, fileOptions.replaceExisting)
       .done(function (file) {
         io.writeTextAsync(file, TodoList.serializeTask(task))
           .done(function () {
             TodoList.alert("All done!");
             TodoList.populateTaskList();
             TodoList.cancelTaskClick(); // Очистка UI после сохранения
           })
       },
       function (error) {
         TodoList.alert("Unable to create file. Sorry about that!");
     });
```

Именем файла служит описание задания, а папкой, где файл создается или хранится, является для приложения роуминговая папка. Следует заметить, что файлы при совпадении имен молча (без выдачи запроса) перезаписываются. Это означает, что у вас не может быть двух заданий с одним и тем же описанием.

Также следует заметить, что после сохранения задания пользователь получает подтверждение, а список заданий автоматически обновляется, пополняясь только что созданным или обновленным заданием (рис. 10.12).

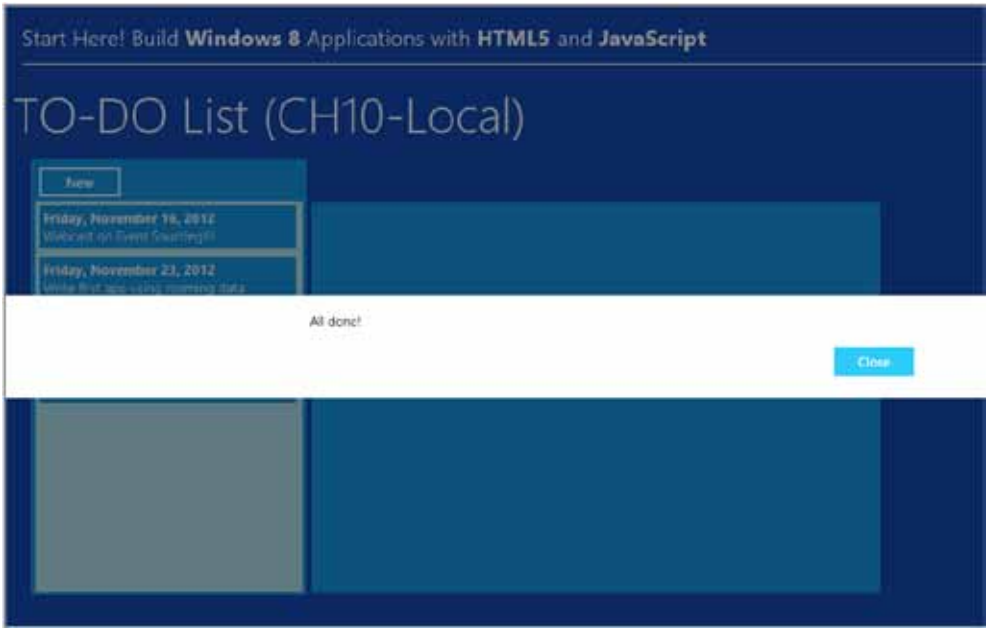

Рис. 10.12. Задание успешно создано

Сразу после успешного создания задания представление списка обновляется, а только что созданное задание убирается из редактора. Следующим шагом является выбор одной из существующих в списке позиций и ее последующее редактирование.

Выбор позиции для редактирования

В главе 7 рассказывалось, как обработать выбор позиции в компоненте `ListView`. Сейчас нам тоже нужно выбрать задание в списке и полностью показать его в редакторе. То есть к компоненту `ListView` добавляем обработчик выбора позиции. Для этого требуется новый метод в файле todolist.js:

```
TodoList.setupTaskList = function () {
    var listview = document.getElementById("task-listview").winControl;
    listview.itemTemplate = document.getElementById("task-listitem");
    listview.addEventListener("iteminvoked", TodoList.taskSelected);
}
```

В этом методе есть несколько строк, уже знакомых нам по методу `TodoList.populateTaskList`. В частности, он берет на себя работу по конфигурированию (например, связывание шаблонов и событий), которая должна выполняться лишь однажды.

Метод `TodoList.setupTaskList` вызывается внутри метода `TodoList.init` непосредственно перед вызовом метода `TodoList.populateTaskList`:

```
TodoList.setupTaskList();
TodoList.populateTaskList();
```

Разумеется, из метода `TodoList.populateTaskList` нужно удалить две строки, которые перешли в метод `TodoList.setupTaskList`.

И наконец, нужно позаботиться о коде, который запускается, когда пользователь выбирает задание в списке. Для этого в файл todolist.js нужно добавить следующий метод:

```
TodoList.taskSelected = function (eventInfo) {
   eventInfo.detail.itemPromise.then(function (item) {
     TodoList.displayTask(item.data);
   });
}
```

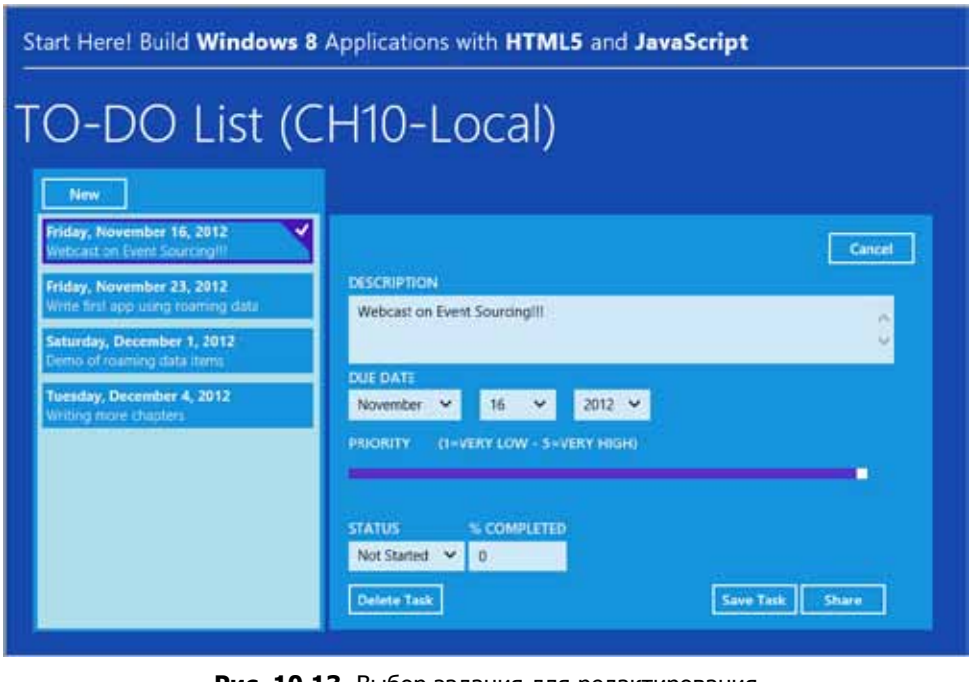

Рис. 10.13. Выбор задания для редактирования

В нем запрашивается выбранная позиция (в асинхронном режиме), а после того как она будет получена, извлекаются содержащиеся в ней данные — то есть задание

Task, которое передается уже знакомому методу TodoList.displayTask. Полученный результат показан на рис. 10.13.

Еще один дополнительный шаг, который вы при желании можете сделать, состоит в том, чтобы после закрытия текущего задания на компоненте ListView не было выбранных позиций. Пользователи закрывают задание щелчком на кнопке Cancel (Отмена). Новая версия метода cancelTaskClick имеет следующий вид:

```
TodoList.cancelTaskClick = function () {
    // Скрытие редактора
    var editor = document.getElementById("editor-container");
    editor.style.display = "none";

    // Удаление любого выделения с представления списка
    var listview = document.getElementById("task-listview").winControl;
    listview.selection.clear();
}
```

Последнее, что осталось сделать в упражнении, — реализовать удаление существующего задания.

Удаление заданий

На рис. 10.13 можно увидеть новую кнопку — Delete Task (Удалить задание). При наличии локальной или роуминговой папки пользователи не могут полностью или частично удалить весь набор созданных ими данных. Чтобы избавиться от данных, достаточно удалить приложение. Поэтому вполне разумно, что приложение предоставляет ряд элементов пользовательского интерфейса, призванных помочь пользователям удалять любые нежелательные фрагменты данных. Разметка для кнопки Delete Task (Удалить задачу):

```
<div style="float:left">
    <button id="buttonDeleteTask">Delete Task</button>
</div>
```

Эту кнопку нужно поместить рядом с кнопками Add Task (Добавить задание) и Share (Обмен данными), а используемый атрибут style выровняет элемент по левому краю контейнера. Вам также нужно зарегистрировать обработчик события click, добавив к методу TodoList.init следующую строку:

```
document.getElementById("buttonDeleteTask")
  .addEventListener("click", TodoList.deleteTaskClick);
```

Затем нужно заняться реализацией метода TodoList.deleteTaskClick. Реализация этого метода разбивается на две части, сначала запрашивается подтверждение, а затем производится удаление файла, относящегося к текущему открытому заданию. Следующий код формирует окно сообщения с двумя кнопками. Кнопка, которой подтверждается операция, вызывает метод TodoList.deleteTask (его мы вскоре создадим). А кнопка отмены просто завершает текущую операцию:

```
TodoList.deleteTaskClick = function () {
   var message = "Are you sure you want to delete the task?";
   var msg = new Windows.UI.Popups.MessageDialog(message);
   msg.commands.append
 (new Windows.UI.Popups.UICommand("Yes, proceed!", TodoList.deleteTask));
   msg.commands.append
 (new Windows.UI.Popups.UICommand("No, I'm not sure...", function() {}));
   msg.defaultCommandIndex = 1;
   msg.showAsync();
}
```

Свойство `defaultCommandIndex` устанавливает индекс с нулевым начальным значением для той кнопки, которая выбирается по умолчанию. В данном случае, как показано на рис. 10.14, выбирается кнопка No (Нет).

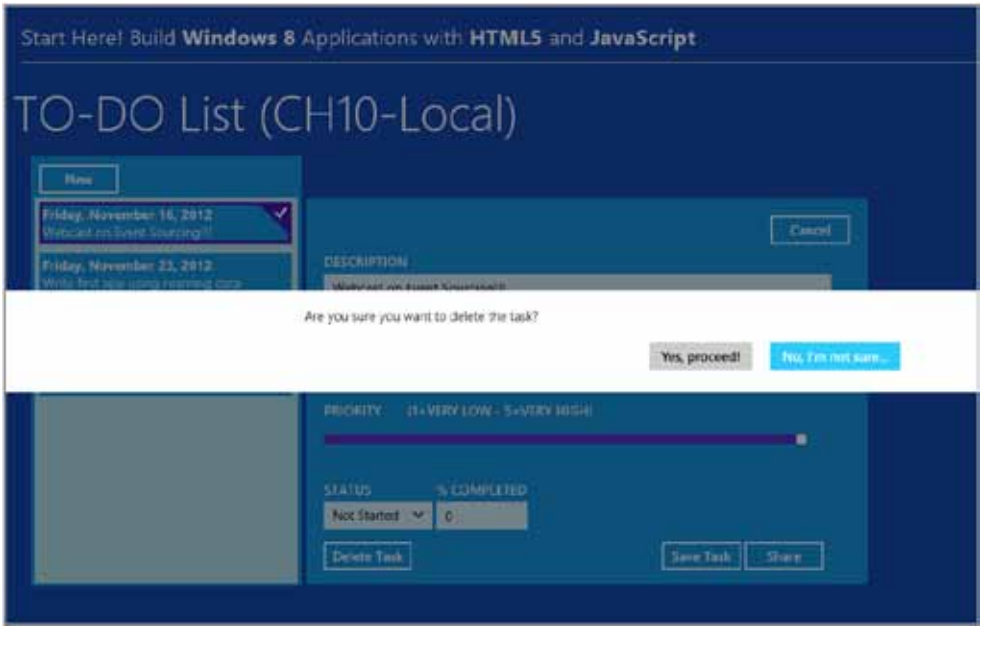

Рис. 10.14. Удаление задачи

Код удаления файла выглядит так:

```
TodoList.deleteTask = function () {
   // Получение задания
   var task = TodoList.getTaskFromUI();

   // Определение местонахождения файла и удаление найденного файла
   var name = task.description;
   var localFolder = Windows.Storage.ApplicationData.current.roamingFolder;
   localFolder.getFileAsync(name)
     .then(function (file) {
```

```
        file.deleteAsync().then(function () {
        TodoList.cancelTaskClick();
        TodoList.populateTaskList();
      });
    });
}
```

Сначала мы получаем объект, который относится к заданию, выводимому в данный момент в редакторе, а затем мы получаем имя соответствующего файла — в данном случае имя файла совпадает с описанием задания. И наконец, определяется местонахождение файла в роуминговой папке. Это делается методом `getFileAsync`, который возвращает ссылку на единственный файл, если таковой существует. Для удаления файла служит метод `deleteAsync`. После этого редактор очищается, а представление списка обновляется, чтобы из него исчезло удаленное задание.

Выводы

Использовать файлы необходимо любому серьезному приложению. Приложения Магазина Windows не являются исключением. В данной главе мы изучили основные операции с файлами и познакомились с различными вариантами хранилищ, доступными приложениям. Особый интерес для приложений Магазина Windows представляют роуминговые папка и параметры. Хранение данных в роуминговой папке и (или) пользовательских параметров в словаре параметров, доступном посредством роуминга, дает возможность операционной системе синхронизировать эти данные с облаком, чтобы ими могла воспользоваться другая копия приложения, запущенная на других устройствах (под управлением той же учетной записи Windows).

Если это обстоятельство вас еще не впечатлило, обратите внимание на ряд довольно симпатичных рекламных роликов об операционной системе iOS, которые стали появляться не так уж давно. В этих рекламных роликах пользователь, находясь в поезде, читает в iPad нужную ему страницу электронной книги. А когда он возвращается домой и удобно устраивается перед компьютером Mac, то запускает то же самое приложение и как по мановенью волшебной палочки оказывается на той же самой странице, на которой прервал свое чтение. Воспользовавшись роуминговыми папкой и параметрами, вы получите точно такое же волшебство.

Следующей задачей, которую нужно решить, чтобы сложить пазл, составляющий приложение магазина Windows, является получение удаленных данных через Интернет. Именно это и является темой следующей главы.

11 Работа с удаленными данными

Любая мелочь, но осмысленная, значит в жизни гораздо больше, чем все самое грандиозное, но бессмысленное.

Карл Юнг

За редким исключением все мобильные приложения относятся к одной из следующих категорий: приложения, которые не могут работать без интернет-подключения, и приложения, которые при отсутствии такого подключения частично сохраняют свою работоспособность. Нельзя считать себя по-настоящему хорошим разработчиком приложений для Microsoft Windows 8, не умея работать с удаленными данными через Интернет.

В этой главе мы научимся работать с данными, загружаемыми из удаленного источника по протоколу HTTP. Сначала мы узнаем, как выполнить HTTP-вызов для загрузки данных. Затем перед нами будет поставлена более сложная задача интерпретации загруженных данных с целью их использования в Windows-приложении.

В первом упражнении мы рассмотрим пример получения и вывода на экран новостного канала с общедоступного сайта. Мы загрузим RSS-канал и преобразуем формат данных, чтобы их можно было визуализировать в компоненте представления списка. Во втором упражнении мы выясним, как загрузить JSON-данные с сайта Flickr и как их встроить в динамически создаваемый макет.

Работа с RSS-данными

Многие веб-сайты делятся своим контентом посредством размеченного текста, доступного для загрузки с общедоступных и упомянутых в документации URL-адресов, с данными, отформатированными, как правило, в виде RSS-канала. Этот канал, по сути, является текстом в формате XML, организованным в соответствии с конкретной схемой. Формат RSS (Really Simple Syndication — очень простое получение информации) чаще всего применяют для публикации часто обновляемого контента, например блогов, новостных заголовков и ссылок на элементы мультимедийных галерей в стандартном формате. Обычный RSS-канал содержит сводку данных, доступную на исходном сайте, плюс какую-нибудь дополнительную информацию, например дату публикации, имя автора и ссылку на полный контент.

Получение удаленных данных

Начинать нужно с создания нового проекта Windows 8, используя для этого шаблон Blank App (Пустое приложение). Затем нужно создать новую папку Pages и добавить знакомые нам по предыдущим главам файлы header.html и footer.html, после чего соответствующим образом отредактировать файлы default.html и default.css. (Это нужно сделать только для того, чтобы обеспечить единство внешнего вида всех приложений.) Но важнее открыть файл default.js и добавить следующий код начальной загрузки:

```
app.onready = function (args) {
   rssReaderApp.init();
};
```

Функция `init` вызывается, когда приложение полностью загружено и готово реагировать на команды пользователя. Однако самой функции `init` у нас пока нет. Чтобы ее создать, добавьте новый JavaScript-файл в проект, назовите его rssReaderApp.js и поместите в папку Jsfolder. На данном этапе у этого файла должно быть следующее содержимое:

```
var rssReaderApp = rssReaderApp || {};
 rssReaderApp.init = function () {
   // Все, что нужно сделать
}
```

Теперь у нас есть работоспособное, но пустое приложение. Давайте его немного оживим.

Знакомство с XHR

В Windows 8 для доступа к удаленным конечным HTTP-точкам используется объект `WinJS.xhr`. Этот объект дает возможность запросить внешний URL-адрес и вернуть

контент, предоставляемый удаленным сервером. Как мы вскоре узнаем, объект `WinJS.xhr` поддерживает разнообразные варианты настройки, но в большинстве случаев ему для работы достаточно передать обычный HTTP-адрес, по которому он должен обратиться.

Изменим содержимое функции `rssReaderApp.init`:

```
// Это URL-адрес, с которого предоставляется RSS-контент
 rssReaderApp.Feed = "http://news.google.com/news?pz=1&output=rss";
 rssReaderApp.init = function () {

 WinJS.xhr({ url: rssReaderApp.Feed }).then(function (rss) {
   // Проведение каких-нибудь операций с данными
   });
}
```

Сначала мы сохраняем URL-адрес, к которому нам нужен доступ, в качестве открытого члена глобального объекта `rssReaderApp`. Выбор URL-адреса, возвращающего RSS-данные, возлагается на вас. Если вы еще не знакомы с RSS-контентом, его можно узнать по известному значку, который можно встретить на многих вебсайтах (рис. 11.1).

Рис. 11.1. Популярный значок, идентифицирующий ссылки на URL-адреса, возвращающие RSS-данные

После определения вызываемого URL-адреса он передается объекту `WinJS.xhr`. Как и всегда с потенциально продолжительными операциями, код нужно писать в асинхронной форме:

```
WinJS.xhr({ url: rssReaderApp.Feed }).then( ... );
```

Объект `WinJS.xhr` передается со свойством `url`, имеющим значение вызываемого URL-адреса. На этом требования заканчиваются. Для вызова больше ничего не нужно. В метод `then` помещается код, который запускается после получения ответа от сервера с вызываемым URL-адресом, в нашем случае — после получения RSS-данных. (Код метода `then` обработает данные и подготовит их к выводу на экран.)

Только что вы прочитали, что единственным предварительным условием для вызова `WinJS.xhr` является наличие URL-адреса. Но все же следует помнить, что правильно указанный URL-адрес не является гарантией успеха. Вызов может окончиться

неудачей по нескольким причинам, и вы должны уметь выявлять сбои. Кроме того, нужно убедиться, что Windows 8 позволяет выходить за границы локальной машины и получать доступ в Интернет. И наконец, вы должны убедиться, что формат отправленного запроса подходит серверу, получившему этот запрос. Давайте рассмотрим варианты дальнейшего конфигурирования запроса.

Конфигурирование объекта WinJS.XHR

В табл. 11.1 перечислены параметры, которые могут быть дополнительно связаны с вызовом объекта `WinJS.xhr`. Эти параметры позволяют задать типа запроса (например, GET или POST), указать пользовательские учетные данные для удаленной аутентификации, установить HTTP-заголовки, и т. д.

Таблица 11.1. Параметры вызова объекта WinJS.xhr

Параметр	Описание
url	Обязательный параметр, задающий вызываемый URL-адрес. URL-адрес может быть абсолютным или относительным. Также поддерживается протокол HTTPS
type	HTTP-метод, используемый для открытия соединения с указанным URL-адресом. Допустимыми значениями для этого параметра являются GET, POST, PUT, DELETE или HEAD. Регистр букв значения не имеет. Если этот параметр не задан, по умолчанию для него устанавливается значение GET
user password	Эти параметры могут использоваться для указания учетных данных, проверяемых на целевом устройстве перед обслуживанием запроса. Если параметр user пропущен или пуст, а сайт требует аутентификации, пользователь увидит окно входа в систему. Если параметр user пропущен или пуст, параметр password игнорируется
headers	Этот параметр можно настроить на JavaScript-объект, в именах свойств которого указываются допустимые HTTP-заголовки. Затем в качестве значений свойств устанавливаются значения заголовков в HTTP-запросе
data	Этот параметр обозначает JavaScript-объект, содержащий данные, обычно передаваемые серверу через запрос POST
responseType	Указывает тип ожидаемого ответа от запроса GET. Допустимыми значениями являются text (используется по умолчанию), json, blob (для двоичного контента) и document (для XML-объекта)

Объект `WinJS.xhr` возвращает JavaScript-объект `Promise` (обязательство), позволяющий разработчику без особого труда подготовить любые шаги, следующие за асинхронной HTTP-операцией. И хотя при работе с JavaScript-обязательствами и асинхронными операциями в предыдущих главах мы уже использовали функции `then` и `done`, будет полезно еще раз вспомнить о том, чем они отличаются друг от друга.

Асинхронные операции библиотеки WinJS (JavaScript Windows 8) всегда возвращают объект `Promise`. Как следует из имени этого объекта, он представляет собой обязательство предоставить какие-нибудь полезные данные в ближайшем будущем. Поэтому разработчику позволяется указать следующие шаги, используя такие функции, как `Promise.done` и `Promise.then`. Обе функции обозначают некое действие, выполняемое сразу же, как только обещанные данные станут доступными. Так чем же они отличаются друг от друга?

Функции `Promise.done` и `Promise.then` совершенно одинаковы за исключением того, что вызов `Promise.done` разрывает цепочку инструкций, возвращая вместо объекта `Promise` значение `undefined`. Иными словами, можно написать следующий код:

```
WinJS.xhr(url).then(doThis).then(doThat).done(doAlsoThis);
```

Но нельзя написать такой код:

```
WinJS.xhr(url).done(doThis).then(doThat);
```

То есть функция `done` должна быть последней в цепочке инструкций.

Обработка ошибок при работе с HTTP-запросами

HTTP-запрос может быть не выполнен по ряду причин. Например, соединение может быть разорвано до выполнения запроса, или же запрос может быть отклонен сервером из-за неправильного оформления или отсутствия нужных учетных данных. И наконец, запрос может «зависнуть», превысив заданное время ожидания. Как же справиться со всеми этими ситуациями? Именно здесь вступают в игру расширенные возможности JavaScript-объектов `Promise`.

У функции `done`, как и у функции `then`, существует следующий прототип:

```
WinJS.xhr({ url: ... }).then(
   function completed(request) {
     // Запрос завершен успешно
   },
   function error(error) {
     // Запрос по каким-то причинам не выполнен
   }
});
```

До сих пор функции `then` объекта обязательства передавалась только одна функция. При успешном завершении запроса функция `then` вызывала первую переданную функцию. Функция `completed` получала ответ от сервера в качестве своего единственного аргумента.

Но, как видно из прототипа, объекту обязательства можно передать и вторую функцию.

Эта вторая функция является для запроса обработчиком ошибок. Если запрос по какой-нибудь причине не выполнен, функция обработки ошибки вызывается

системой автоматически. Функция `error` получает объект `error`, имеющий такие свойства, как `status`, `statusText` и `message`. Следует заметить, что в зависимости от характера ошибки значения для некоторых из этих свойств могут быть не установлены. Можно воспользоваться универсальным сообщением, указанным в следующем коде (рис. 11.2):

```
WinJS.xhr({ url: rssReaderApp.Feed }).then(
    function (response) {
      rssReaderApp.parseFeed(response);
    },
    function (error) {
      rssReaderApp.alert("A download error occurred.");
    }
);
```

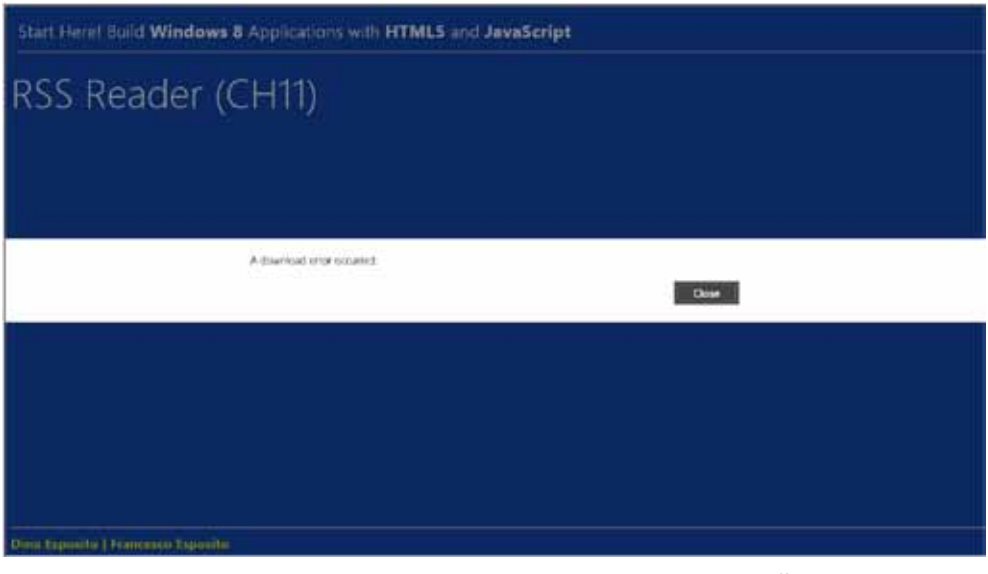

Рис. 11.2. Обработка ошибок в ходе HTTP-операций

Задание времени ожидания запросов

Запрос на удаленный сервер может выполняться некоторое время. Время ответа зависит от ряда факторов, включая ширину полосы пропускания и объем трафика на целевом сайте. Если для вашего приложения важную роль играет фиксированное время, может потребоваться связать время ожидания с вашим объектом `WinJS.xhr` запроса. Если это сделать, то при незавершенности запроса по прошествии указанного времени запрос автоматически отменяется и выдается ошибка, передающаяся обратно вызывающему приложению. Для установки времени ожидания (таймаута) нужен следующий код:

```
WinJS.Promise.timeout(3000,
   WinJS.xhr({ url: ... }).then(
     function (response) {
       // Обработка ответа
     },
     function (error) {
       rssReaderApp.alert(error.message);
     })
);
```

В общем, вся работа сводится к заключению вызова `WinJS.xhr` в оболочку из вызова `WinJS.Promise.timeout`. С помощью первого параметра устанавливается время ожидания (в миллисекундах), а вторым параметром служит вызов `WinJS.xhr`. Заметьте, что если запрос не выполняется из-за истечения времени ожидания, то в свойстве `message` получаемого объекта ошибки указывается универсальное сообщение, которое можно напрямую вывести на экран — в данном случае это сообщение Canceled (рис. 11.3).

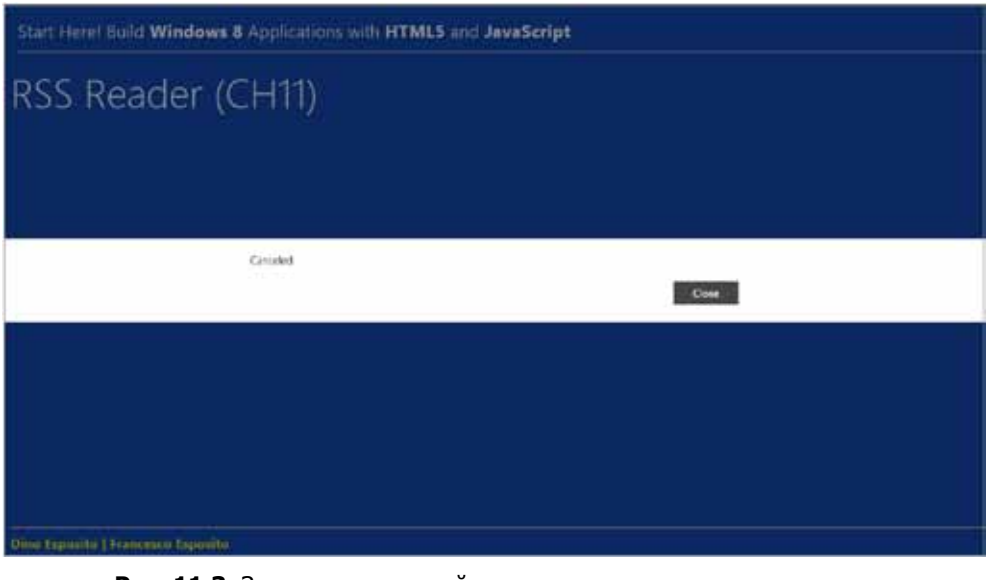

Рис. 11.3. Запрос, отмененный по превышению времени ожидания

Просмотр манифеста приложения

Во все проекты приложений Магазина Windows включается файл манифеста. До сих пор потребности в манифесте у нас не возникало. Однако если ваше приложение нуждается в особых возможностях, например в возможности создавать HTTP-запросы, отправляемые через Интернет, манифест может понадобиться. Давайте взглянем на файл манифеста.

Найдите в папке своего проекта файл package.appxmanifest и откройте его. Затем перейдите на вкладку Capabilities (Возможности). Если сделать это применительно к учебному проекту RssReader, разрабатываемому в данной главе, экран должен выглядеть так, как показано на рис. 11.4.

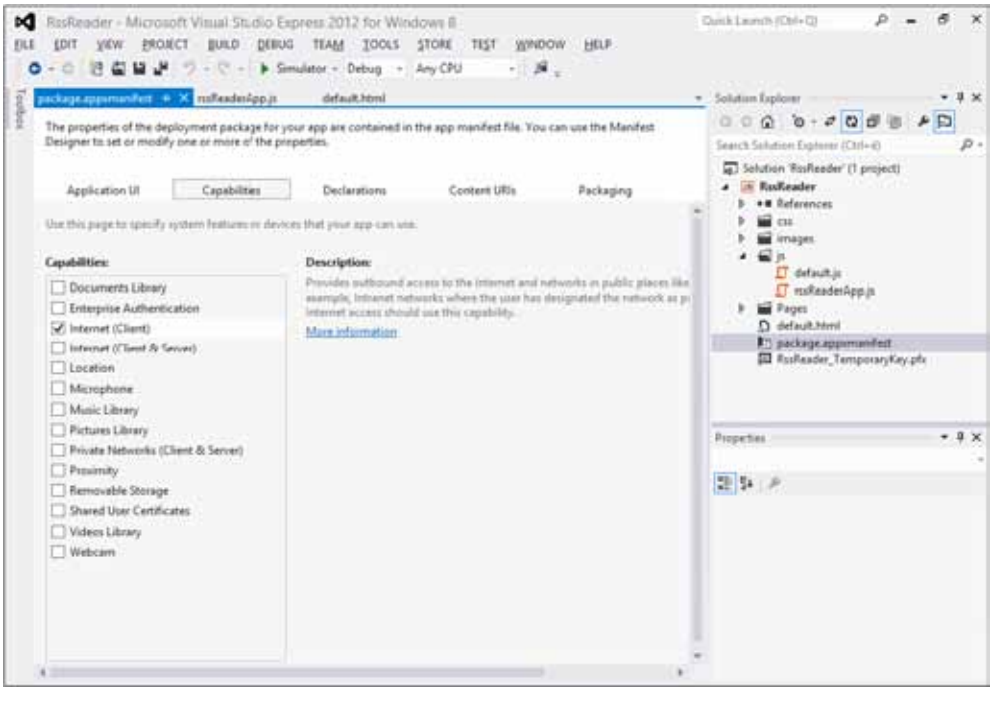

Рис. 11.4. Файл манифеста для учебного проекта

Каждая из перечисленных возможностей доступна в Windows 8. Однако здесь важно понять, что, если та или иная возможность не помечена флажком, система времени выполнения Windows 8 не даст вашему приложение вызвать соответствующий прикладной программный интерфейс. Как показано на рис. 11.4, для того чтобы объект `WinJS.xhr` мог работать должным образом, нужно установить флажок Internet Client (Интернет-клиент).

Теперь попробуйте сбросить флажок Internet Client (Интернет-клиент) и запустить учебное приложение RssReader. Приложение будет успешно скомпилировано и развернуто на устройстве, но никакого видимого эффекта не произойдет — никаких сообщений об ошибках, никаких аварийных завершений, никаких неудач. Почему? Все очень просто, система времени выполнения Windows 8 блокирует все вызовы, для выполнения которых требуется четко заявленная возможность.

ВНИМАНИЕ

Из этого следует вынести следующий урок: если программируемая функция не работает без видимых причин, убедитесь, что вы заявили соответствующую возможность, требуемую вашей функции. Также учтите, что в Microsoft Visual Studio 2012 возможность интернет-клиента (Internet Client) является минимально необходимой для работы WinJS.xhr и единственной, включенной по умолчанию.

Синтаксический разбор загруженных данных и их вывод на экран

Теперь вы уже знаете все, что нужно для успешной отправки вызова на удаленную конечную HTTP-точку и получения оттуда неких данных. Следующий шаг заключается в том, чтобы сделать с этими данными что-нибудь полезное. В данном упражнении мы обратимся к службе Google News, чтобы загрузить последние новости в формате RSS-канала. Затем мы произведем синтаксический разбор возвращенной строки, преобразовав ее в список новостей, и используем его, чтобы заполнить представление списка.

Расширение приложения чтения RSS-канала

Перед тем как заняться синтаксическим разбором RSS-данных, нужно внести в пользовательский интерфейс приложения некоторые изменения. Сначала откройте файл default.html и введите следующую разметку:

```
<h1>RSS Reader (CH11)</h1>
    <div id="newslist" data-win-control="WinJS.UI.ListView">
    </div>
  <div id="splitView">
    <div id="titleDetail"></div>
      <div id="pubDateDetail"></div>
      <div id="categoryDetail"></div>
    <div id="descriptionDetail"></div>
</div>
```

Элемент `div` с идентификатором `newslist` является компонентом `ListView`, который будет содержать все загруженные новости. А элемент `div` с идентификатором `splitView` предназначен для предварительного просмотра выбранной новости.

Еще одной важной частью добавляемой разметки является HTML-шаблон для новостей. Но перед тем, как создавать этот шаблон, у вас должно быть четкое представление о том, что мы получаем из конечной HTTP-точки и как превратить все это в полезные данные.

Синтаксический разбор RSS-контента

Откройте файл rssReaderApp.js и отредактируйте функцию `rssReaderApp.init`:

```
WinJS.xhr({ url: rssReaderApp.Feed }).then(
    function (response) {
      rssReaderApp.parseFeed(response);
    },
    function (error) {
      rssReaderApp.alert("A download error occurred.");
    }
);
```

Функция `rssReaderApp.parseFeed` получит контент с удаленного URL-адреса и предпримет попытку преобразовать его в формат, подходящий для приложения. Следует понимать, что при неблагоприятном развитии событий в ходе загрузки управление передается функции обработки ошибок, поэтому, если вызов функции `rssReaderApp.parseFeed` действительно происходит, значит, данные для обработки имеются.

Самое главное для функции `rssReaderApp.parseFeed` — убедиться, что получаемые данные имеют формат, который она в состоянии обработать. Поскольку загружаются RSS-данные, которые относятся к формату XML, проверку этого факта лучше выполнить в начальной части функции `parseFeed`:

```
rssReaderApp.parseFeed = function (response) {
    if (response.responseXML == null) {
      rssReaderApp.alert("Invalid data");
      return;
    }

    // Сюда помещается код обработки данных ...
}
```

Ответ, получаемый с сервера, заключается в объект, передаваемый функции. Если получаемые данные могут быть визуализированы в виде объектной модели XML-документа, свойство `responseXML` не будет иметь значение `null` и управление можно передавать коду, который мы напишем для запроса различных RSS-элементов.

RSS-канал соответствует следующей схеме:

```
<rss ...>
    <channel ...>
      <item>
        <title> ... </title>
  <link> ... </link>
  <guid> ... </guid >
  <description> ... </description >
  <category> ... </category>
      </item>

      ...
    </channel>
</rss>
```

В системе синтаксического разбора (в парсере) сначала нужно выбрать все элементы `item`, а затем осуществить их последовательный перебор для извлечения конкретной информации, такой как заголовок, описание и ссылка на источник,

а также, возможно, категория. Следующий код позволяет выбрать все элементы `item` из всего XML-документа:

```
var items = response.responseXML.querySelectorAll("rss > channel > item");
```

Синтаксис запроса практически идентичен синтаксису CSS-запроса, который рассматривался в главе 3; он означает следующее: «дай мне все элементы по имени `item`, являющиеся дочерними для элементов `rss` и `channel`». Затем нужно создать цикл `for-each` и отдельно обработать содержимое каждого элемента `item`. В следующем коде в качестве примера показано извлечение заголовка записи первой опубликованной новости:

```
var title = items[0].querySelector("title").textContent;
```

Поскольку речь идет о синтаксическом разборе, нужно в заключение решить проблему сохранения данных после разбора. В идеале можно собрать информацию в массиве, с которым легче будет работать и который затем можно будет привязать к представлению списка. Вот окончательная версия кода для синтаксического разбора:

```
rssReaderApp.parseFeed = function (response) {
   if (response.responseXML == null) {
     rssReaderApp.alert("Invalid data");
     return;
   }
   var items = response.responseXML.querySelectorAll("rss > channel > item");
   for (var n = 0; n < items.length; n++) {
     var newsElement = {};
     newsElement.title = items[n].querySelector("title").textContent;
     newsElement.link = items[n].querySelector("link").textContent;
     newsElement.guid = items[n].querySelector("guid").textContent;
     newsElement.pubDate = items[n].querySelector("pubDate").textContent;
     newsElement.description =
           items[n].querySelector("description").textContent;

     // Проверка категории
     var category = "[No category]";
     if (items[n].querySelector("category") != null)
       category = items[n].querySelector("category").textContent;
     newsElement.category = category;

     RssReader.Items.push(newsElement);
   }
}
```

Здесь нужно обратить внимание на две вещи. Во-первых, если работа ведется с одним конкретным RSS-каналом, вы можете адаптировать свой код для ожидаемого формата. Если же вы планируете создать некий универсальный компонент для чтения RSS-канала, который пользователи смогут настраивать для получения данных из разных источников, нужно заняться каждым фрагментом данных, необходимым для доступа. Например, в канале Google News можно найти узел категории, но

многие другие RSS-каналы его игнорируют. Чтобы избежать исключений времени выполнения, нужно перед попыткой считывания информации программным путем проверять наличие узла категории. Чтобы обезопасить себя, может также понадобиться дополнить проверку, распространив ее на все узлы, которые будут считываться из RSS-канала. Ведь никогда не угадаешь, что на самом деле будет получено от веб-серверов!

Во-вторых, в предыдущем коде нужно обратить внимание на коллекцию `RssReader.Items`, в которой хранятся все новости, прошедшие синтаксический разбор. В упражнениях предыдущих глав привязка данных всегда осуществлялась программным путем. Иными словами, всегда создавался код сценария для привязки коллекции данных к объекту `ListView` или к другим компонентам пользовательского интерфейса, допускающим привязку.

Но в этом упражнении мы поступим по-другому. Привязку данных можно также осуществить декларативно при разработке приложения. Для этого используется объект `RssReader`. Этот объект определяется в самом начале файла rssReaderApp.js:

```
WinJS.Namespace.define("RssReader", { Items: new WinJS.Binding.List() });
```

Объект `RssReader` имеет свойство `Items`, инициализированное пустым списком привязки. При выполнении цикла система синтаксического разбора просто добавляет новости в коллекцию `Items` объекта `RSSReader`. Следовательно, объект `RssReader.Items` станет источником данных для компонента `ListView`, созданного нами в пользовательском интерфейсе.

ПРИМЕЧАНИЕ

Независимо от того, программно или декларативно осуществляется привязка данных, на функциональность это не влияет. Выбор между программным и декларативным способом зависит в основном от личных предпочтений разработчика.

Отображение записи RSS-канала на пользовательский интерфейс

Теперь все готово к отображению новостного контента на HTML-шаблон, с помощью которого записи будут визуализироваться в представлении списка (в компоненте `ListView`). Учебный HTML-шаблон выглядит следующим образом:

```
<div id="newsItemTemplate" data-win-control="WinJS.Binding.Template"
style="display: none;">
   <div class="listItem">
     <div id="titleDiv" data-win-bind="innerText: title"></div>
     <div id="categoryDiv" data-win-bind="innerText: category"></div>
     <div id="pubDateDiv" data-win-bind="innerText: pubDate"></div>
   </div>
</div>
```

Как видите, шаблон получает данные из свойств `title`, `category` и `pubDate` связанного объекта новостей. Теперь уже почти все сделано. Завершающий шаг состоит из привязки самого компонента `ListView` к его источнику данных. Эта привязка осуществляется в файле default.html через атрибут `data-win-options` декларативным способом:

```
<div id="newslist"
     data-win-control="WinJS.UI.ListView"

     data-win-options="{ itemDataSource: RssReader.Items.dataSource,
                         itemTemplate: newsItemTemplate,
                         layout: {type: WinJS.UI.ListLayout} }">
</div>
```

Теперь, если откомпилировать и запустить учебный проект, можно будет увидеть на экране нечто подобное показанному на рис. 11.5.

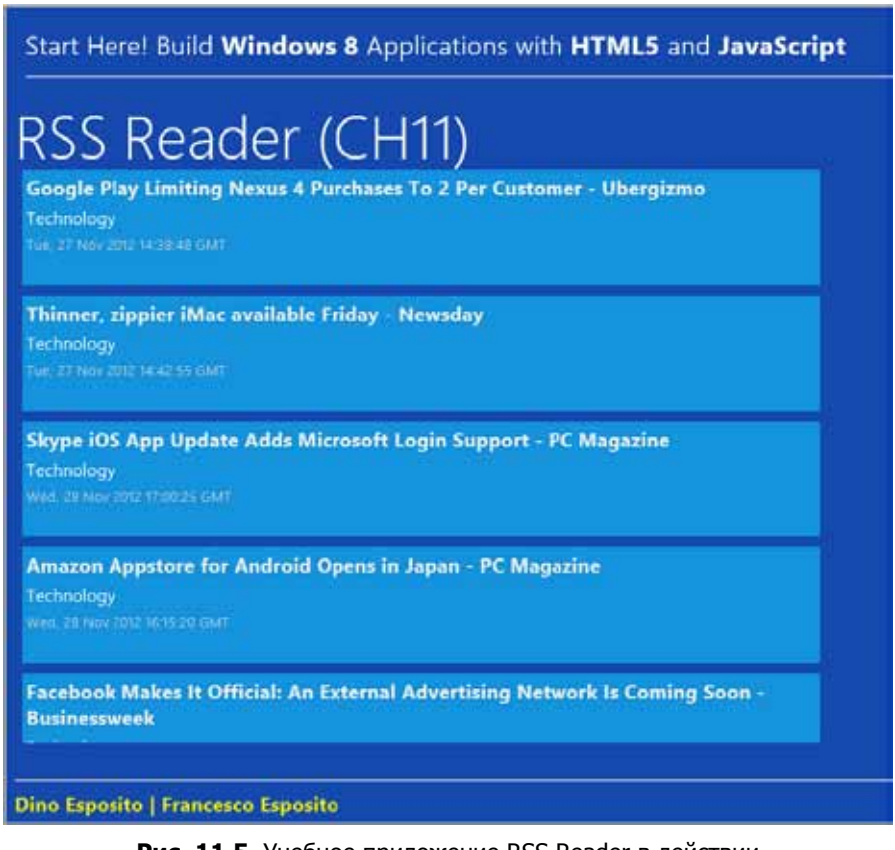

Рис. 11.5. Учебное приложение RSS Reader в действии

Углубление в данные

В предыдущих главах мы выяснили, как сделать так, чтобы каждая позиция в представлении списка (в компоненте `ListView`) реагировала на выбор пользователя, давая ему возможность получать дополнительную информацию в процессе такого «углубления» в данные. Как показано на рис. 11.5, на данном этапе пользователь может видеть только заголовок, публикуемые данные и категорию. А как предоставить пользователю возможность предварительного просмотра новостного контента? Или возможность перехода к реальному источнику контента? Конечно, можно было бы добавить дополнительные элементы пользовательского интерфейса и привязать их к представлению списка, но для этого потребовалось бы дополнительное экранное пространство. Вместо этого можно разрешить пользователям углубляться в данные, когда им требуется дополнительная информация. А для этого нужно сделать так, чтобы каждая позиция в представлении списка реагировала на щелчок, а затем придумать способ вывода на экран, как минимум, поля описания загруженных новостей.

Наделение позиций в списке новостей способностью реагировать на щелчок

Если вы выполняли все упражнения в предыдущих главах, то знаете, что нужно сделать в первую очередь, чтобы взять под контроль пользовательские щелчки на выводимых на экран позициях списка новостей. Чтобы дать возможность непосредственного выбора заданного поведения и разрешить одиночный выбор, нужно изменить некоторые параметры компонента `ListView`:

```
<div id="newslist"
    data-win-control="WinJS.UI.ListView"
    data-win-options="{ itemDataSource: RssReader.Items.dataSource,
                        itemTemplate: newsItemTemplate,
                        layout: {type: WinJS.UI.ListLayout},
                        selectionMode: 'single',
                        tapBehavior: 'directSelect' }">
</div>
```

Затем нужно в файле rssReaderApp.js зарегистрировать обработчик события `itemInvoked` объекта `ListView`:

```
rssReaderApp.init = function () {
    var listview = document.getElementById("newslist");
    listview.addEventListener("iteminvoked", rssReaderApp.preview);

    // Весь остальной код помещается здесь
    ...
}
```

И наконец, мы добрались до кода, который выводит на экран описание новостей.

Визуализация необработанной HTML-разметки

В зависимости от полученного RSS-канала в качестве описания новостей может быть получен как простой текст, так и более выразительная HTML-разметка. Можно было бы убрать из контента потенциально опасные элементы, удалив и (или) экранировав все встречающиеся в тексте HTML-теги. Для этого потребовалось бы передать описание в функцию, возвращающую очищенную строку. В качестве альтернативы можно выводить на экран всю HTML-разметку, особенно если есть платформа, такая как Windows 8, которая дает вам полный доступ к визуальным возможностям веб-браузера. Главное, что нужно иметь в виду при выборе первого или второго варианта, — это степень доверия к тому или иному RSS-провайдеру, а также к общему уровню безопасности платформы. Если, к примеру, вами создается веб-сайт, получающий данные из источника, предоставляемого пользователем, то вас следует предупредить заранее: не визуализируйте необработанную HTML-разметку! Если же данные поступают от хорошо известного RSS-провайдера в контексте приложения для Windows 8, то визуализировать потенциально опасную HTML-разметку, конечно же, можно. Вот код для предварительного просмотра выбранных новостей:

```
rssReaderApp.preview = function (e) {
    var index = e.detail.itemIndex;
    var currentArticle = RssReader.Items.getAt(index);
    var amended = currentArticle.description;
    amended = amended.replace("src=" + '"//', "src=" + '"http://');
    document.getElementById("titleDetail").innerHTML = currentArticle.title;
    document.getElementById("pubDateDetail").innerHTML =
      currentArticle.pubDate;
    document.getElementById("categoryDetail").innerHTML =
      currentArticle.category;
    document.getElementById("descriptionDetail").innerHTML = amended;
}
```

Сначала извлекается индекс выбранной в списке позиции и новости, к которым она относится. В этом месте устанавливается свойство `innerHTML` HTML-элемента, который предназначен для описания новости, а также для другой контекстной информации, такой как дата публикации и категория. При предоставлении возможности предварительного просмотра новости, нужно настроить элементы, находящиеся в элементе `div` с идентификатором `splitView`, который мы создали ранее в данном упражнении.

Описания новостей, которые берутся из канала Google News, обычно содержат теги `img` (это теги изображений). По каким-то причинам в URL-адресах этих изображений нет префикса `http://`, что не дает Windows 8 возможности визуализировать изображение. Как показано в предыдущем коде, все можно исправить с помощью обычной операции замены строки. Конечный результат показан на рис. 11.6.

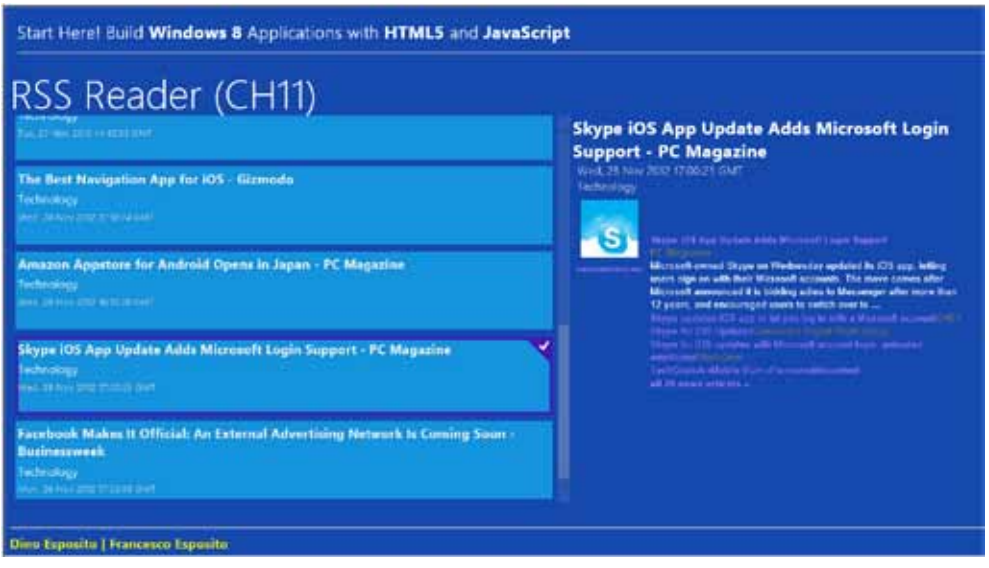

Рис. 11.6. Окончательная версия приложения RSS Reader с областью предварительного просмотра выбранной новости

Работа с JSON-данными

Еще один весьма распространенный сценарий для приложений Магазина Windows относится к загрузке данных с сервисов, которые выставляют данные не в формате RSS, а в формате JSON (JavaScript Object Notation). Обычно веб-сервисы предлагают свой контент в нескольких форматах, самыми распространенными из которых являются XML (и, в частности, RSS) и JSON. В предыдущей главе нам уже приходилось работать с прикладным программным интерфейсом для формата JSON. Там мы сохраняли данные приложения в формате JSON, а чуть позже считывали их обратно. При этом мы целиком и полностью отвечали за весь цикл работы с данными — сериализацию, сохранение и десериализацию.

Однако когда вы получаете JSON-данные из удаленного источника, то можете управлять только той частью всего цикла, которая отвечает за десериализацию. К сожалению, это не означает, что вам достаточно сделать только половину работы. Хотя JSON является широко используемым и де-факто стандартным форматом, он не относится к области точных наук. Как будет показано в следующем упражнении, существует риск получения испорченных JSON-данных. Когда это случается, исправлять ситуацию приходится самостоятельно.

Проектирование системы просмотра фотографий с сайта Flickr

В этом упражнении мы создадим приложение, получающее и показывающее общедоступные фотографии с популярного веб-сайта Flickr. Ссылки на фотографии и сопутствующая информация, такая как сведения об авторе и описание, загружаются в виде JSON-строки, а затем визуализируются с помощью компонента `ListView`.

Создание приложения FlickrPhotoViewer

Создайте новый проект на основе шаблона Blank App (Пустое приложение) и назовите его FlickrPhotoViewer. После добавления новой папки Pages с файлами header.html и footer.html (как это делалось в предыдущих упражнениях), откройте файл default.js и добавьте обычный обработчик события `onready`. Использовав данный метод, вы сможете, как показано далее, управлять инициализацией приложения:

```
app.onready = function (args) {
   flickrApp.init();
};
```

Следующим шагом является добавление к проекту нового JavaScript-файла flickrApp.js. Изначально в этом файле содержится следующий код:

```
var flickrApp = flickrApp || {};
 flickrApp.init = function () {
   // Здесь будет дополнительный код
}
```

Теперь обратите внимание на пользовательский интерфейс приложения и откройте файл default.html. В нем должна быть такая разметка:

```
<!DOCTYPE html>
 <html>
 <head>
   <meta charset="utf-8" />
   <title>Flickr Photo Viewer</title>

   <!-- Ссылки WinJS -->
   <link href="//Microsoft.WinJS.1.0/css/ui-dark.css" rel="stylesheet" />
   <script src="//Microsoft.WinJS.1.0/js/base.js"></script>
   <script src="//Microsoft.WinJS.1.0/js/ui.js"></script>

   <!-- Ссылки Flickr Photo Viewer -->
   <link href="/css/default.css" rel="stylesheet" />
   <script src="/js/default.js"></script>
   <script src="/js/flickrApp.js"></script>
 </head>

 <body>
   <div data-win-control="WinJS.UI.HtmlControl"
      data-win-options="{uri:'/pages/header.html'}"></div>
   <h1>Flickr Viewer (CH11)</h1>
```

```
    <br />
    <input type="text" placeholder="Subject" id="subject" value="" />
    <button id="buttonSearch">Search</button>

    <div id="picturesList" data-win-control="WinJS.UI.ListView">
    </div>
    <div data-win-control="WinJS.UI.HtmlControl"
       data-win-options="{uri:'/pages/footer.html'}"></div>
  </body>
</html>
```

Кроме обычной разметки страница содержит компонент `ListView`, привязанный к элементу `div` с идентификатором `picturesList`. Чтобы разрешить пользователям вашего приложения набирать ключевые слова для выбора соответствующих изображений, нужно добавить поле ввода текста и кнопку.

Для кнопки Search (Поиск) нужен обработчик щелчка. Добавьте его в метод `flickrApp.init`:

```
flickrApp.init = function () {
    document.getElementById("buttonSearch")
    .addEventListener("click", flickrApp.searchClick);
}
```

Обработчик щелчка будет отвечать за отправку HTTP-вызова веб-сервису Flickr и за обработку полученных в ответ JSON-данных.

Просмотр открытого Flickr-канала

В данном примере будет осуществляться доступ к открытому Flickr-каналу фотографий, который не требует ни аутентификации, ни каких-либо специальных Flickr-идентификаторов приложения. Вам нужно лишь знать URL-адрес, по которому осуществляется вызов. Как и в первом упражнении этой главы, этот адрес сохраняется в качестве члена глобального объекта `flickrApp`:

```
flickrApp.Source = "http://api.flickr.com/services/feeds/photos_public.gne"
                 + "?tagmode=any&format=json&nojsoncallback=1&tags='{0}'";
```

Краткое пояснение параметров URL-адреса дано в табл. 11.2.

Таблица 11.2. Параметры строки запроса к открытому веб-сервису Flickr

Параметр	Описание
tagmode	У этого параметра может быть значение all или any; он указывает, должны ли любые выбранные изображения соответствовать всем или только любому из указанных тегов
format	Если нужно получить ответ в формате JSON, для этого параметра нужно установить значение json. Можно также указать, что вызов должен вернуть ответ в формате RSS или в других форматах, полное описание которых можно найти по адресу http://www.flickr.com/services/feeds/docs/photos_public

продолжение ⇩

Таблица 11.2 (*продолжение*)

Параметр	Описание
nojsoncallback	В приложении Магазина Windows этот параметр является обязательным и должен иметь значение 1. Если Flickr API вызывается из JavaScript-приложения, то этот параметр не нужен. Он относится к тому, как среда хоста фактически выполняет вызов удаленного сайта
tags	В качестве значения этого параметра можно указать список ключевых слов, разделенных запятыми, которые должны присутствовать в описаниях искомых фотографий

Следует заметить, что для свойства `flickrApp.Source` устанавливается строковое значение, содержащее символ-заменитель `{0}`. В .NET-программировании этот тип записи используется для динамического форматирования строк. Замысел состоит в том, что при щелчке пользователя на кнопке Search (Поиск) обработчик прочитает набранные теги и вставит их в строку URL-адреса вместо символа-заменителя.

Запись `{0}` обычна в .NET-программировании для Windows, но не в JavaScript-программировании. Поэтому нужно создать вспомогательную функцию, осуществляющую замену. Она интересна тем, что демонстрирует эффективность JavaScript-технологии — управление прототипами объектов.

Подготовка URL-адреса для Flickr

Желательно создать еще один JavaScript-файл, который называется helpers.js. Сделайте ссылку на этот файл в своем файле default.html и убедитесь в том, что ссылка на файл helpers.js предшествует ссылке на файл flickrApp.js.

Исходное содержимое файла helpers.js:

```
////////////////////////////////////////////////////////////
 // Применение .NET-соглашения {n} к форматированию строк
 //
 String.prototype.format = function () {
   var theString = this,
     count = arguments.length;

   while (count--) {
     theString = theString.replace(
       new RegExp('\\{' + count + '\\}', 'gm'), arguments[count]);
   };

   return theString;
};
```

Код добавляет функцию `format` к прототипу «родного» JavaScript-объекта `String`. В результате дополнительный метод `format` видит любую строку, обрабатываемую в области видимости файла helpers.js. После ссылки на файл helpers.js можно написать следующий код:

```
flickrApp.searchClick = function () {
   var tags = document.getElementById("subject").value;
   flickrApp.download(tags);
 }
 flickrApp.download = function (tags) {
   // Добавленте тегов к URL-адресу Flickr
   var url = flickrApp.Source.format(tags);

   // Получение фотографий
 WinJS.xhr({ url: url }).then(function (json) {
   // Здесь производится синтаксический разбор JSON-канала
   });
}
```

Обработчик события `click` кнопки Search получает любой текст, набранный в поле ввода, и передает его дальше только что созданной функции `download`. Внутри этой функции составляется URL-адрес, по которому методом `WinJS.xhr` осуществляется вызов. Учтите, что, если передать пустой список тегов, в ответ все равно будут получены фотографии. Дело в том, что строгого требования по проверке переменной `tags` на пустое значение или на `null` не предъявляется.

Получение данных в формате JSON

Функция `WinJS.xhr` служит для загрузки JSON-строки, описывающей выбранные фотографии. Функция `WinJS.xhr` работает так же, как и в предыдущем упражнении. Вам нужно определить две функции: одну, которая выполняется в случае успешной загрузки, и другую, которая выполняется при отклонении запроса, и передать их методу `then` в объекте обязательства, возвращаемом функцией `WinJS.xhr`:

```
WinJS.xhr({ url: url }).then(
   function (response) {
     flickrApp.parseFeed(response.responseText);
   },
   function (response) {
     var message = "Error downloading photos.";
     if (response.message != null)
       message += response.message;
     flickrApp.alert(message);
   }
);
```

Метод `then` передает своим функциям объект ответа, полученный из удаленного источника. Чтобы получить обычный текст, содержащийся в теле JSON-ответа, нужно запросить свойство `responseText`.

Сердце нашего приложения находится в теле метода `flickrApp.parseFeed`. Его реализация выглядит следующим образом:

```
flickrApp.parseFeed = function (json) {
   var pictures = JSON.parse(json);
   for (var i = 0; i < pictures.items.length; i++) {
```

продолжение ⇩

```
      var pictureElement = {};
      pictureElement.photoUrl = pictures.items[i].media.m;

      // Привязка к listview
      ...
    }
}
```

Получаемая JSON-строка имеет следующий формат:

```
{
   "title": "Recent Uploads tagged tennis",
   "link": "http://www.flickr.com/photos/tags/tennis/",
   "description": "",
   "modified": "2012-11-30T07:56:42Z",
   "generator": "http://www.flickr.com/",
   "items": [
     {
       "title": "...",
       "link": "http://www.flickr.com/photos/craftydogma/8232121888/",
       "media": {
         "m": http://farm9.staticflickr.com/8350/8232121888_9f762d7e5e_m.jpg
       },
       "date_taken": "2012-05-05T19:04:48-08:00",
       "description": "<p> ... </p>",
       "published": "2012-11-30T07: 56: 42Z",
       "author": "...",
       "author_id": "...",
       "tags": "tennis vintage photo"
     }
   ]
}
```

Чтобы получить доступ к URL-адресу фотографии требуется следующее выражение:

```
pictureElement.photoUrl = pictures.items[i].media.m
```

URL-адрес фотографии, а также другую информацию, такую как дату и описание, можно поместить в удобную структуру данных — в объект `pictureElement`.

Добавление изображений к пользовательскому интерфейсу

Чтобы вывести полученные изображения на экран, можно воспользоваться уже хорошо знакомым нам представлением списка — компонентом `ListView`. В файле default.html уже имеется элемент `div`, связанный с компонентом `ListView`. Следующий шаг состоит в привязке компонента `ListView` к источнику данных и определении шаблона для позиции списка.

Поскольку мы просто хотим выводить изображения на экран, можно воспользоваться следующим несложным HTML-шаблоном:

```
<div id="picturesItemTemplate"
   data-win-control="WinJS.Binding.Template" style="display: none;">
   <div class="listItem">
     <img id="picture" data-win-bind="src: photoUrl" alt="" />
   </div>
</div>
```

Атрибут `src` элемента `img` привязан к свойству `photoUrl` объекта данных. Шаблон добавляется в начало тела страницы в файле default.html.

Точно так же, как это делалось в первом упражнении данной главы, объект списка привязки определяется в верхней части файла flickrApp.js:

```
WinJS.Namespace.define(
  "FlickrFeed", { Pictures: new WinJS.Binding.List() });
```

Этот список заполняется в процессе перебора JSON-контента:

```
flickrApp.parseFeed = function (json) {
   var pictures = JSON.parse(json);
   for (var i = 0; i < pictures.items.length; i++) {
     var pictureElement = {};
     pictureElement.photoUrl = pictures.items[i].media.m;

     // Добавление объекта к listview
     FlickrFeed.Pictures.push(pictureElement);
   }
}
```

И наконец, компонент `ListView` привязывается к списку `FlickrFeed.Pictures`. Для этого в файл default.html нужно добавить следующую разметку:

```
<div id="picturesList" data-win-control="WinJS.UI.ListView"
   data-win-options="{ itemDataSource: FlickrFeed.Pictures.dataSource,
               itemTemplate: picturesItemTemplate,
               layout: {type: WinJS.UI.GridLayout} }">
</div>
```

На рис. 11.7 показано, как будет выглядеть приложение при удачном стечении событий.

Рис. 11.7. Приложение просмотра фотографий с сайта Flickr в действии

Совершенствование приложения

Как уже упоминалось, формат JSON не имеет отношения к точным наукам. Поэтому не исключается возможность получения недопустимой JSON-строки. Если метод `JSON.parse` не сможет выполнить синтаксический разбор загруженного текста, ваше приложение просто выбросит исключение, как показано на рис. 11.8.

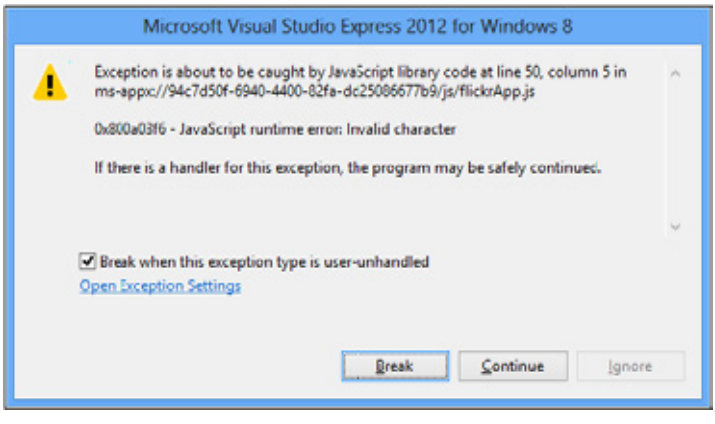

Рис. 11.8. Результат работы приложения с недопустимым JSON-контентом

Обработка недопустимого JSON-контента

При обнаружении недопустимого JSON-контента сразу выяснить причину проблемы невозможно. Известно лишь то, что система синтаксического разбора дала сбой, но в чем его причина? Может быть, причиной ошибки являются проблемы в самой системе? Или причина кроется в недопустимом JSON-контенте? При создании приложения для работы с сайтом Flickr окно ошибки, показанное на рис. 11.8, появлялось у меня неоднократно, причем зачастую без видимой причины. Иногда, просто изменив несколько тегов, можно было восстановить работоспособность приложения.

Когда система синтаксического разбора жалуется на недопустимый JSON-контент, сначала нужно убедиться в его допустимости. Как показано на рис. 11.9, JSON-текст можно проверить, скопировав его в соответствующий редактор по адресу http://jsonlint.com.

Но как заполучить из ответа JSON-текст, чтобы его можно было вставить в редактор на сайте JSONLint?

В Visual Studio нужно поставить контрольную точку на ту строку, в которой осуществляется синтаксический разбор контента (рис. 11.10). Контрольная точка приведет к остановке приложения на этой строке, предоставив вам возможность изучить информационный контекст времени выполнения.

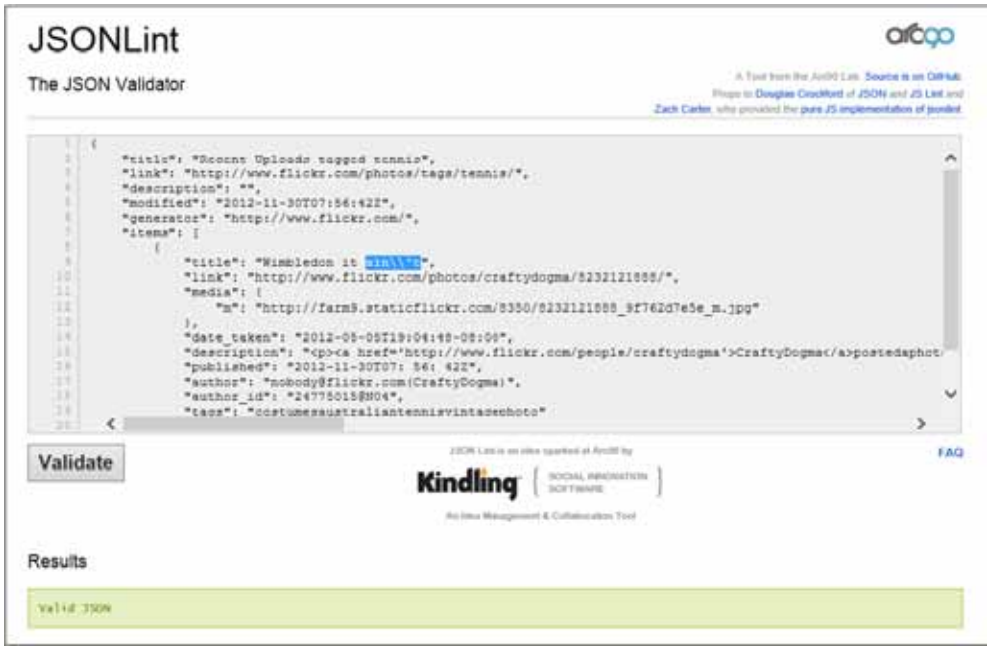

Рис. 11.9. Веб-сайт JSONLint

Рис. 11.10. Установка контрольной точки с целью изучения содержимого переменной json

Когда выполнение будет приостановлено на контрольной точке, нужно подвести указатель мыши к переменной `json`. Появится всплывающая подсказка, возможно, слишком широкая. Если текста слишком много, изучение его в подсказке может стать непростым занятием. В качестве альтернативы можно щелкнуть на значке стрелки непосредственно перед текстом и выбрать нужный нам режим — в данном случае Text visualizer (Визуализатор текста), как показано на рис. 11.11.

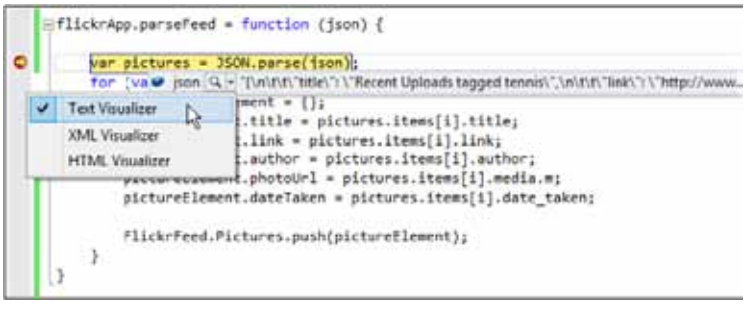

Рис. 11.11. Изучение содержимого переменной json в удобном визуализаторе

Визуализатор текста представляет собой простой текстовый редактор, в котором удобно просматривать содержимое переменной `json`, то есть тот текст, который предстоит передать JSON-системе синтаксического разбора и который, как известно, и стал причиной возникновения исключения Invalid character (Недопустимый символ). Вы можете просто выделить этот текст и скопировать его в буфер обмена. Для этого можно либо нажать комбинацию клавиш Ctrl+C, либо щелкнуть на тексте правой кнопкой мыши и выбрать команду Copy (Копировать), как показано на рис. 11.12.

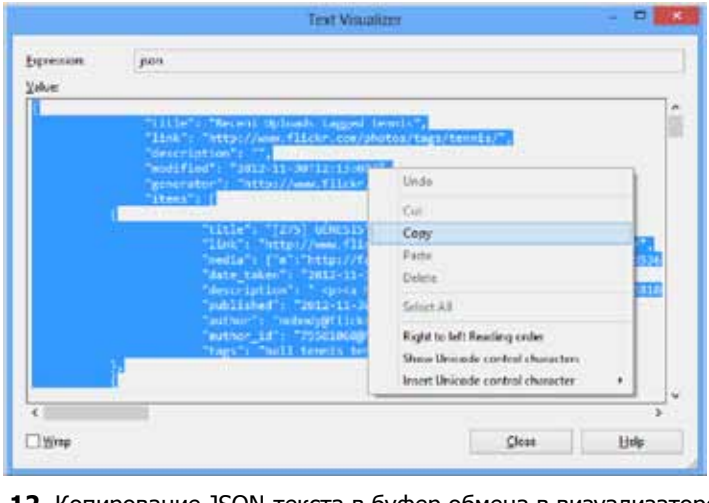

Рис. 11.12. Копирование JSON-текста в буфер обмена в визуализаторе текста

Имея в буфере обмена скопированный JSON-текст, можно перейти на упомянутый ранее веб-сайт JSONLint, провести синтаксический разбор текста в редакторе и проверить его на допустимость.

Исправление недопустимого JSON-контента

Оказывается, веб-сайт Flickr не всегда возвращает допустимый JSON-контент, который может успешно пройти проверку на сайте JSONLint. В то же время любой JSON-контент, прошедший проверку на сайте JSONLint, столь же успешно проходит синтаксический разбор с помощью функции `JSON.parse` в Windows 8.

Одна из проблем, с которыми вы можете столкнуться, заключается в том, что Flickr не всегда правильно обрабатывает одинарную кавычку, встречающуюся в некоторых описаниях. Рассмотрим, к примеру, следующий текст:

```
Wimbledon it ain\'t
```

Этот текст можно встретить в описаниях некоторых фотографий. Чтобы он прошел JSON-проверку, перед одинарной кавычкой нужно поставить второй экранирующий обратный слэш:

```
Wimbledon it ain\\'t
```

Решить эту проблему можно быстро и эффективно, введя в прототип `String` еще одно расширение. Откройте файл helpers.js еще раз и добавьте в него следующий код:

```
String.prototype.doubleEscapeSingleQuotes = function () {
   var theString = this;
   if (theString != null && theString != "") {
     return theString.replace(/\'/g, "\\'");
   } else {
     return theString;
   }
}
```

Важно заметить, что простой вызов функции `replace` работать не будет, поскольку эта функция заменяет соответствующую строку только при ее первом появлении. Нужно, как в показанном примере, заключить заменяемую строку в регулярное выражение и обязательно добавить квалификатор `g`, чтобы работа велась со всей строкой:

```
/Заменяемая_строка/g
```

Тогда код для синтаксического разбора JSON-контента приобретет следующий вид:

```
flickrApp.parseFeed = function (json) {
   var amendedText = json.doubleEscapeSingleQuotes();
   var pictures = JSON.parse(amendedText);
   for (var i = 0; i < pictures.items.length; i++) {
     var pictureElement = {};
     pictureElement.photoUrl = pictures.items[i].media.m;
```

продолжение ⇙

```
      // Добавление объекта к listview
      FlickrFeed.Pictures.push(pictureElement);
    }
}
```

После добавления этого кода можно корректно проводить разбор JSON-текста.

Вывод на экран разных наборов фотографий

Чтобы завершить упражнение, можно добавить вторую кнопку для очистки страницы от текущих фотографий. Создаваемая функция также пригодится для тихого удаления представления, когда пользователь приступает к новому поиску. Следует заметить, что без шага «очищения» при любом новом поиске фотографии будут просто добавляться к существующему списку.

Чтобы добавить кнопку Clear (Очистить), в файл default.html нужно включить следующую разметку:

```
<button id="buttonClear">Clear</button>
```

Затем в методе `flickrApp.init` нужно зарегистрировать обработчик события щелчка на кнопке:

```
flickrApp.init = function () {
    document.getElementById("buttonSearch")
            .addEventListener("click", flickrApp.searchClick);
    document.getElementById("buttonClear")
            .addEventListener("click", flickrApp.clearClick);
}
```

И наконец, остается только создать код для обработчика и для очистки фотографий:

```
flickrApp.clearClick = function () {
    flickrApp.clear();
 }
 flickrApp.clear = function () {
    // Обнуление свойства длины списка
    FlickrFeed.Pictures.splice(0, FlickrFeed.Pictures.length);
}
```

Вызов метода `flickrApp.clear` можно также поместить в функцию загрузки непосредственно перед удаленным вызовом:

```
flickrApp.download = function (tags) {
 // Добавление тегов в URL-адрес Flickr
 var url = flickrApp.Source.format(tags);

 // Очистка имеющихся фотографий (если они есть)
 flickrApp.clear();

 // Получение новых фотографий
 WinJS.xhr({ url: url }).then( ... );
};
```

Эта финальная версия позволяет многократно выполнять поиск, после чего показывать только самые последние найденные фотографии, что удобнее для пользователя и лучше с точки зрения общей производительности приложения.

Выводы

Возможность получения данных из удаленного источника по протоколу HTTP является ключевой практически для всех современных приложений и особенно для приложений, развернутых на мобильных устройствах. В приложении Магазина Windows, написанного средствами HTML и JavaScript, HTTP-доступ осуществляется посредством объекта `WinJS.xhr`. В данной главе мы сделали два упражнения, касающихся создания HTTP-запросов для получения удаленных данных и использования ответов, содержащих данные в форматах RSS и JSON. Хотя в данной главе не ставилась цель охватить все возможные сценарии, наиболее часто встречающиеся из них мы рассмотрели.

Эта глава завершает ту часть книги, которая посвящена основным аспектам программирования для Windows 8. В оставшейся части мы займемся более сложными вещами, такими как устройства и сенсоры, живые плитки и публикация. Следующая глава посвящена программированию для встроенных устройств и датчиков, в том числе GPS и веб-камер.

12 Доступ к устройствам и датчикам

Если вы человек увлеченный, вам никогда не придется заниматься поиском новых интересов. Они придут к вам сами. Если вы искренне чем-то увлечены, ваше увлечение непременно приведет вас к чему-то еще.

Элеонора Рузвельт

Своим успехом смартфоны и планшетные устройства обязаны, главным образом, высокому качеству своих датчиков и внутренних устройств. Если вернуться к первым дням iPhone, можно вспомнить, как люди были удивлены сейчас уже вполне обычному приложению Flashlight. По сути, это было игровое приложение, хотя и с некоторой практической пользой. Ассортимент устройств и датчиков для таких современных изделий, как планшетные компьютеры, оснащенные операционной системой Microsoft Windows 8, настолько богат, что дает возможность разработчикам придумывать и создавать совершенно новые типы приложений.

Как разработчик программ для Windows 8, вы даже из среды программирования на базе JavaScript имеете доступ к разнообразным прикладным программным интерфейсам среды выполнения Windows, которые позволяют управлять широким спектром датчиков, включая GPS, датчик освещенности, акселерометр и компас. Кроме того, приложения Магазина Windows могут получать доступ к устройствам, как подключенным к компьютеру, так и встроенным в него. Типичными примерами являются принтеры и веб-камеры.

В данной главе мы научимся программно управлять веб-камерой, распечатывать информацию и работать с одним из наиболее полезных датчиков — системой GPS, возвращающей текущее местоположение пользователя.

Работа с веб-камерой

Сегодня почти все компьютеры и устройства поставляются с веб-камерами высокого разрешения. Веб-камеры обычно используются встроенными программами для поддержки видеоконференций и видеочатов. Кроме того, приложения с их помощью позволяют получать моментальные фотоснимки пользователя. С точки зрения программирования веб-камера ничем не отличается от других компонентов оборудования. Разработчик изучает открытый прикладной программный интерфейс, предназначенный для работы с веб-камерой, реализует корректные вызовы и обеспечивает ожидаемое поведение этого устройства. В 1990-х годах работа с оборудованием зачастую была весьма трудоемкой и сложной. К счастью, теперь мы живем и программируем в другом столетии! В качестве примера в нашем очередном упражнении мы создадим приложение Магазина Windows, делающее моментальные снимки пользователя.

Захват потока веб-камеры

Как обычно, сначала нужно создать новый проект для Windows 8, воспользовавшись шаблоном Blank App (Пустое приложение). Затем нужно создать новую папку Pages и добавить общие файлы header.html и footer.html, как это уже неоднократно делалось в предыдущих главах. Как описано ранее в этой книге, нужно соответствующим образом отредактировать файлы default.html и default.css. (Это нужно только для того, чтобы у всех приложений был одинаковый внешний вид.)

Предварительная настройка проекта

А теперь займемся более важным делом. Откройте файл default.js и добавьте следующий код самозагрузки:

```
app.onready = function (args) {
   instantPhotoApp.init();
};
```

На данном этапе вы уже должны знать, что функция `init` вызывается после полной загрузки приложения и его готовности к реагированию на команды пользователя. Но функцию `init` нужно создать. Точно так же, как это уже несколько раз делалось в предыдущх упражнениях, добавьте к проекту новый JavaScript-файл instantPhotoApp.js, поместив его в папку js.

Имя JavaScript-файла особой роли не играет, главное, чтобы оно соответствовало выполняемой работе. Давать файлу имя своего любимого животного, наверное, все же не стоит. В соглашении, используемом в данной книге, предполагается, что основной JavaScript-файл приложения получает имя, соответствующее имени проекта с постфиксом «App».

Исходное содержимое JavaScript-файла instantPhotoApp.js должно выглядеть так:

```
var instantPhotoApp = instantPhotoApp || {};
 instantPhotoApp.init = function () {
   // Сюда помещается все, что будет делаться в этой функции
}
```

Теперь у нас есть работоспособное, но пустое приложение. Для настройки пользовательского интерфейса к нему нужно добавить разметку.

Включите в файл default.html следующий фрагмент HTML-кода:

```
<h1>Instant Photo (CH12)</h1>
 <div class="center">
   <img id="imgPhoto" src="/images/nophoto.png" />
   <br />
   <button id="buttonShoot" class="horizontalBtn">Take a picture</button>
   <br />
</div>
```

Приложение для работы с веб-камерой состоит в основном из кнопки, включающей веб-камеру, и элемента `img`, в котором пользователи будут видеть захваченное изображение. Завершающим подготовительным шагом является добавление обработчика щелчка на кнопке Take A Picture (Сделать снимок). Откройте файл instantPhotoApp.js и введите следующий код:

```
instantPhotoApp.init = function () {
   document.getElementById("buttonShoot")
     .addEventListener("click", instantPhotoApp.takePicture);
   };
   instantPhotoApp.takePicture = function () {
     // Сюда помещается остальной код
}
```

Теперь все настроено и настало время заняться прикладным программным интерфейсом веб-камеры.

Подключение возможности использовать веб-камеру

Код не может получить свободный доступ к веб-камере без явного разрешения со стороны пользователя. Поэтому, чтобы дать приложению возможность успешно работать с веб-камерой, сначала нужно декларировать намерение ее использовать.

Как показано на рис. 12.1, нужно дважды щелкнуть на имеющемся в проекте файле манифеста, в появившемся представлении перейти на вкладку Capabilities (Возможности) и установить флажок Webcam (Веб-камера).

Следует заметить, что подключения возможности использования веб-камеры вполне достаточно для создания приложения, делающего мгновенные снимки. Однако если создается приложение, использующее веб-камеру для захвата видеопотока, то может понадобиться также подключение возможности задействовать микрофон, поскольку возможность использования веб-камеры позволяет получать доступ к видеопотоку, но не к аудиопотоку.

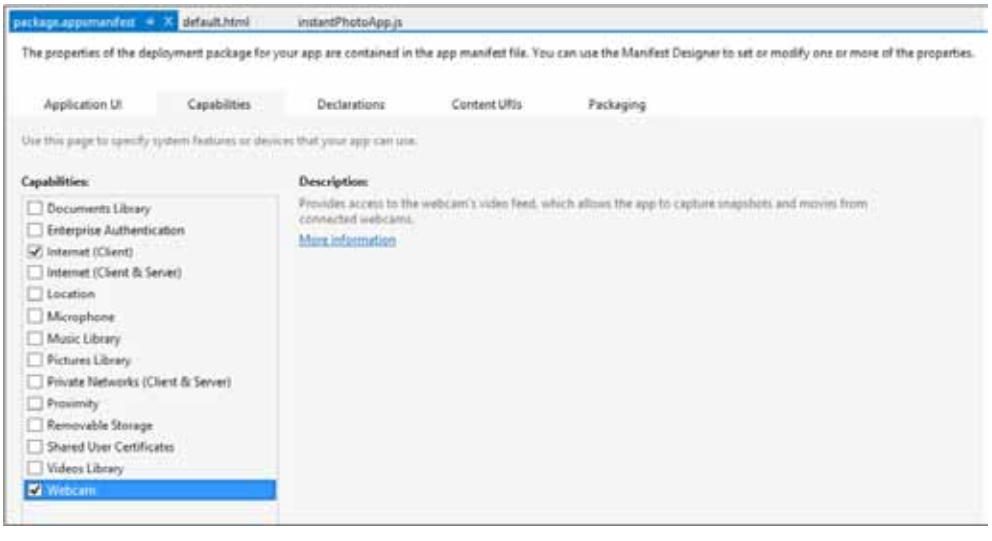

Рис. 12.1. Подключение возможности программного обращения к веб-камере

Настройка веб-камеры

Для работы с веб-камерой в Windows 8 соответствующий прикладной программный интерфейс предоставляет обычное диалоговое окно. Вам остается только получить экземпляр объекта `CameraCaptureUI`, а затем его запустить:

```
var dialog = new Windows.Media.Capture.CameraCaptureUI();
```

Поскольку объект `CameraCaptureUI` работает и с фото, и с видео, он предоставляет два разных свойства для задания соответствующих параметров: `photoSettings` и `videoSettings`. В рамках данного упражнения требуется только свойство `photoSettings`.

Объект `photoSettings` предлагает ряд свойств для задания размера получаемого изображения (в пикселах), соотношения сторон этого изображения и формата файла для сохранения изображения. Чтобы получить соотношение сторон 16:9 и сохранить изображение в формате JPEG, нужен следующий код:

```
var dialog = new Windows.Media.Capture.CameraCaptureUI();
 var aspectRatio = { width: 16, height: 9 };
```

```
 dialog.photoSettings.croppedAspectRatio = aspectRatio;
 dialog.photoSettings.format =
  Windows.Media.Capture.CameraCaptureUIPhotoFormat.jpeg;
```

Если вам не нравится формат JPEG, можно вместо него выбрать формат PNG:

```
dialog.photoSettings.format =
  Windows.Media.Capture.CameraCaptureUIPhotoFormat.png;
```

В общем и целом, формат JPEG позволяет получить более компактный файл, требующий меньше ресурсов и времени на передачу по сети, кроме того, он занимает меньше места при хранении на диске. В отличие от него, формат PNG обеспечивает более высокое качество изображения за счет большего размера файла — обычно PNG-файл примерно вдвое больше JPEG-файла.

Программный доступ к веб-камере

После настройки веб-камеры все готово к открытию диалогового окна, призванного привести пользователя к получению изображения. В следующем листинге показан весь код в файле instantPhotoApp.js, требуемый для захвата изображения, хотя, если быть точным, здесь пока не хватает кода, предназначенного для вывода захваченного изображения на экран:

```
instantPhotoApp.takePicture = function () {
   try {
     var dialog = new Windows.Media.Capture.CameraCaptureUI();
     var aspectRatio = { width: 16, height: 9 };
     dialog.photoSettings.croppedAspectRatio = aspectRatio;
     dialog.photoSettings.format = Windows.Media.Capture.
                 CameraCaptureUIPhotoFormat.jpeg;
     dialog.captureFileAsync(Windows.Media.Capture.CameraCaptureUIMode.photo)
     .then(
       function (file) {
         // Сюда помещается остальной код
       },
       function (err) {
         // Операция прекращена пользователем
       }
     );
   }
   catch (err) {
     // Вывод сообщения об ошибке
   }
}
```

Как видите, ядром операции захвата является метод `captureFileAsync`, предоставляемый объектом `CameraCaptureUI`. Этому методу передается только один аргумент, который управляет тем, что нужно получить от камеры: изображение или видео:

```
dialog.captureFileAsync(
  Windows.Media.Capture.CameraCaptureUIMode.photo).then(...);
```

Метод работает в асинхронном режиме, поэтому для обработки полученного от него результата нужно подготовить JavaScript-обязательство `then` или `done`, получающее

байтовый поток захваченного изображения. Работу приложения иллюстрирует рис. 12.2.

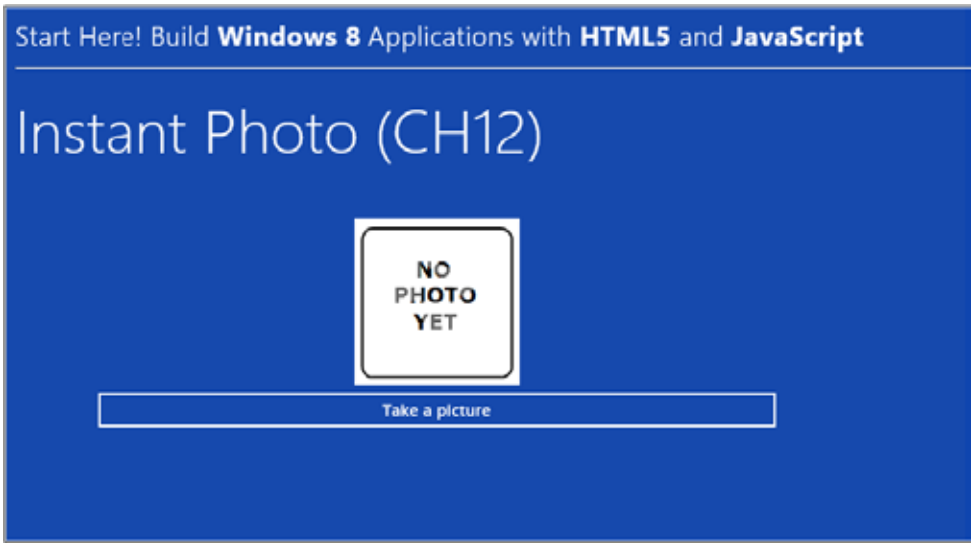

Рис. 12.2. Приложение веб-камеры, готовое к получению изображения

При первом запуске приложения пользователь должен разрешить использование веб-камеры. На рис. 12.3 показан защитный механизм Windows 8 в действии: пользователь должен явным образом разрешить приложению задействовать веб-камеру. Если пользователь не даст разрешения, приложение работать не будет.

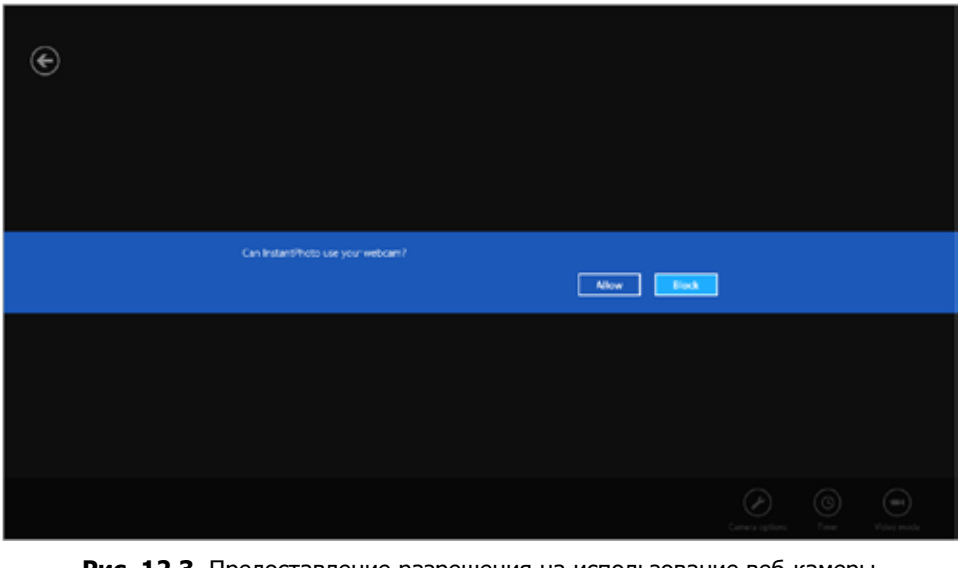

Рис. 12.3. Предоставление разрешения на использование веб-камеры

На рис. 12.4 показана камера после щелчка на кнопке Take A Picture (Сделать снимок).

Рис. 12.4. Веб-камера в действии

Для получения изображения нужно прикоснуться к экрану или дважды щелкнуть кнопкой мыши. Перед этим пользователь может продолжить настройку камеры или даже запустить таймер. Кнопка Back (Назад), расположенная в верхнем левом углу экрана, позволяет вернуться к приложению, не предпринимая никаких действий. При щелчке на кнопке Back (Назад) приложение посчитает операцию выполненной успешно, хотя при этом методы обязательств файл не получает.

После прикосновения, необходимого для получения снимка, диалоговое окно предоставляет возможность кадрировать изображение или сделать еще один снимок, если вам что-то не нравится (рис. 12.5).

ПРИМЕЧАНИЕ

Веб-камера, выводимая на экран в диалоговом окне CameraCaptureUId, является полноэкранной. Помимо нее, Windows 8 предоставляет альтернативный прикладной программный интерфейс, обеспечивающий полный контроль над выходным потоком камеры. С помощью этого усовершенствованного интерфейса можно создавать нестандартные представления и применять фильтры к захваченным данным.

Метод `captureFileAsync` возвращает управление вызвавшему его приложению сразу же после завершения операции захвата изображения. Если снимок был сделан, функция обязательства передает объект, ссылающийся на захваченное изображение. Теперь нужно решить, что делать с захваченным потоком.

Рис. 12.5. Выделение части изображения

Обработка захваченного потока

С потоком, захваченным с веб-камеры, потребуется, наверное, сделать две вещи: вывести изображение на экран, чтобы его мог видеть пользователь, и сохранить изображение в каком-нибудь постоянном месте, например в библиотеке Pictures.

Вывод захваченного изображения

Следующий код делает все, что нужно, для вывода захваченного изображения на экран через элемент `img` главной страницы приложения:

```
dialog.captureFileAsync(Windows.Media.Capture.CameraCaptureUIMode.photo)
   .then(function (file) {
     if (file) {
       var viewer = document.getElementById("imgPhoto");
       viewer.src = URL.createObjectURL(file);
     } else {
       // Захват не состоялся, но без ошибки
     }
  });
```

Для визуализации изображения в элементе `img`, что является единственным известным способом вывода изображений на экран в WinJS-приложении, требуется адрес для создания ссылки, но захваченное изображение возвращается в виде описателя файла из хранилища Windows 8. Необходимо вызвать вспомогательную функцию `URL.createObjectURL`, специально созданную для возвращения URL-адреса,

применяемого для больших двоичных объектов, медиа-объектов или потоков ввода-вывода. На рис. 12.6 показан основной интерфейс приложения после вывода на экран захваченного изображения.

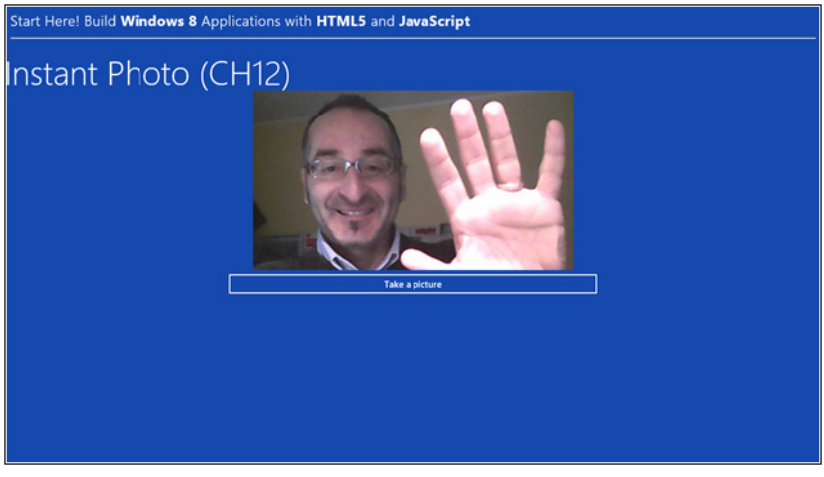

Рис. 12.6. Захваченное изображение в приложении

С выводом захваченного изображения на экран все получилось удачно, но без дополнительного кода при выходе из приложения изображение будет потеряно. Следующим шагом станет небольшое усложнение приложения, позволяющее ему сохранять изображения в хранилище и извлекать их из хранилища.

Добавление возможности использования библиотеки изображений

Приложение Магазина Windows запускается в защищенной среде, которую часто называют *песочницей* (sandbox). В это понятие вкладывается множество смыслов, но применительно к данному упражнению оно означает, что приложение не имеет свободного доступа к любому физическому диску ни по чтению, ни по записи.

В предыдущих главах было показано, что приложение Магазина Windows может сохранять любые данные в собственном зарезервированном пространстве. Для ранее созданного приложения TodoList этого было более чем достаточно, но для учебного приложения Instant Photo, оно, наверное, не подойдет. Выходные данные приложения Instant Photo состоят из контента, к которому пользователь, скорее всего, захочет предоставить доступ другим приложениям.

Для достижения этой цели лучше всего сохранять фотографии в папке Pictures, а для этого требуется заявить еще одну возможность. Поэтому откройте манифест приложения еще раз, перейдите на вкладку Capabilities (Возможности) и установите флажок Pictures Library (Библиотека Pictures), как показано на рис. 12.7.

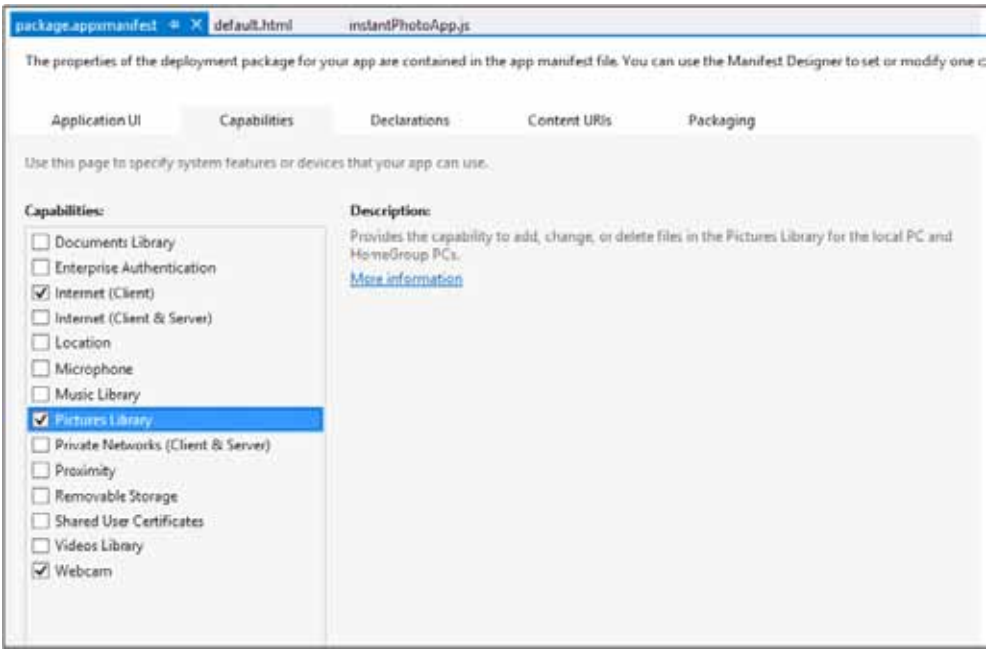

Рис. 12.7. Объявление возможности использования библиотеки изображений

Вы можете решить дать пользователям возможность создать в папке Pictures специальную вложенную папку для хранения всех изображений, полученных ими с помощью приложения Instant Photo. Это можно сделать программно, чем мы сейчас и займемся.

Создание папки для фотографий

Наиболее подходящим временем создания вложенной папки Instant Photo является первый запуск приложения на устройстве. Добавьте к методу `init` в файле instantPhotoApp.js следующий код:

```
var picturesFolder = Windows.Storage.KnownFolders.picturesLibrary;
 picturesFolder.createFolderAsync("Instant Photo",
                 Windows.Storage.CreationCollisionOption.
                 openIfExists)
       .then(function (folder) {
         instantPhotoApp.customFolder = folder;
      });
```

Ссылку на только что созданную папку нужно где-нибудь сохранить, чтобы позже ее можно было бы извлечь при фактическом сохранении изображения. Для этого вам нужно объявить новое глобальное свойство для корневого объекта приложения. Отредактируйте свой файл instantPhotoApp.js, чтобы он начинался следующим кодом:

```
var instantPhotoApp = instantPhotoApp || {};
instantPhotoApp.customFolder = null;
```

Результат выполнения этого кода будет выражаться в том, что при каждом запуске приложения Instant Photo оно будет проверять наличие папки Instant Photo в библиотеке Pictures и создавать ее в случае отсутствия. В любом случае конечный результат будет заключаться в том, что приложение сохранит ссылку на объект папки в только что созданном свойстве `customFolder`.

Сохранение копии изображения

Диалоговое окно захвата данных веб-камеры возвращает ссылку на файл захваченного потока. В терминах API для Windows 8 обязательство `then`, инициированное диалоговым окном камеры, получает объект `StorageFile`. Впервые этот объект попадался нам в главе 10.

У объекта `StorageFile` есть весьма полезный метод `copyAsync`, который копирует файл в заданную папку. Поэтому для сохранения захваченного изображения в библиотеке Pictures нужно добавить в обязательство `then` сразу после кода вывода изображения на экран следующую строку:

```
// Здесь file является объектом хранилища, возвращаемым диалоговым окном
// пользовательского интерфейса захвата изображения с веб-камеры
file.copyAsync(instantPhotoApp.customFolder);
```

Весь код, который нужно выполнить, когда пользователь щелкает на кнопке получения нового снимка, выглядит так:

```
instantPhotoApp.takePicture = function () {
    try {
      var dialog = new Windows.Media.Capture.CameraCaptureUI();
      var aspectRatio = { width: 16, height: 9 };
      dialog.photoSettings.croppedAspectRatio = aspectRatio;
      dialog.photoSettings.format = Windows.Media.Capture.
      CameraCaptureUIPhotoFormat.jpeg;
      dialog.captureFileAsync(Windows.Media.Capture.CameraCaptureUIMode.photo)
        .then(function (file) {
          if (file) {
            var viewer = document.getElementById("imgPhoto");
            viewer.src = URL.createObjectURL(file);
            file.copyAsync(instantPhotoApp.customFolder);
          }
        }
      } catch (err) {
        // Вывод какого-нибудь сообщения об ошибке
      }
}
```

На рис. 12.8 показано несколько фотографий, захваченных, а затем безопасным образом сохраненных в папке Instant Photo библиотеки Pictures.

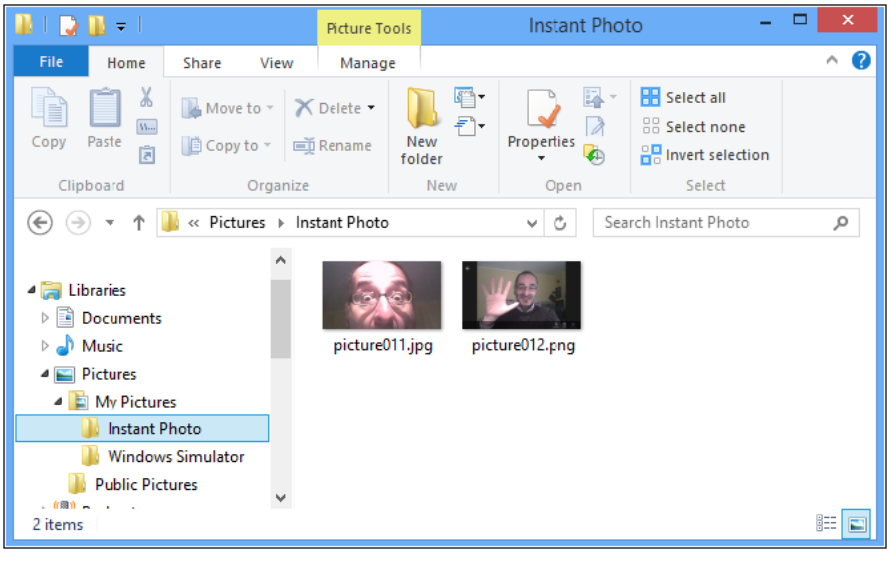

Рис. 12.8. Папка Instant Photo

Работа с принтером

Хотя мир уверенными темпами продвигается к цифровому и виртуальному будущему, печать документов все равно остается важной задачей даже несмотря на экологические проблемы, поднимаемые в печатных изданиях. Для разработчиков вывод на печать еще никогда не был настолько простым, как сейчас. В предыдущие годы разработчикам приходилось воевать с огомным количеством различных драйверов и настроек.

Однако в Windows 8 вывод на печать максимально упростился. Выполнение этой операции из JavaScript-приложения занимает лишь несколько относительно стандартных шагов. Тем не менее вы должны иметь в виду, что JavaScript-приложения не имеют доступа к некоторым нетривиальным возможностям, таким как программное задание нестандартных параметров или печать конкретных страниц.

Что же касается базовых операций, работа с принтером из JavaScript-приложений Магазина Windows осуществляется быстро и просто. Под «базовыми операциями» подразумевается возможность управлять распечатываемым контентом и шаблоном печати, а также возможность вывода на печать контекстных фрагментов данных.

Займемся новым упражнением, которое позволит вам накопить определенный опыт в плане вывода на печать типового контента.

Контракт печати

В Windows 8 вывод на печать регламентируется контрактом печати. То есть чтобы приложение Магазина Windows поддерживало вывод на печать, оно должно зарегистрировать контракт печати. Этот контракт сообщает системе, что приложение поддерживает вывод на печать, в результате, когда пользователь коснется правой стороны экрана, чтобы появилась выдвигающаяся панель, он обнаружит перечень доступных принтеров. Аналогично этому, может понадобиться разрешить пользователю путем, к примеру, щелчка на кнопке отправить задание на печать из интерфейса пользователя. При этом выдвигающаяся панель будет выводиться на экран программным способом, а затем пользователь сможет выбрать принтер и выполнить вывод на печать.

Предварительная настройка учебного приложения

Перед тем как сконцентрироваться на конкретных задачах вывода на печать, нужно настроить учебное приложение. Создайте еще один проект Магазина Windows, воспользовавшись шаблоном Blank App (Пустое приложение), и добавьте папку Pages с файлами header.html и footer.html. Определите в файле default.css обычные CSS-стили. И наконец, добавьте в файл default.js обработчик события `onready` приложения, с помощью которого приложение будет инициализироваться:

```
app.onready = function () {
   printerApp.init();
}
```

Также добавьте в проект новый JavaScript-файл по имени, скажем, printerApp.js, и следующим образом инициализируйте глобальный объект приложения:

```
var printerApp = printerApp || {};
```

Что касается пользовательского интерфейса, для этого приложения больше ничего делать не нужно. Добавьте просто две кнопки для вывода на печать различного контента. Вставьте сразу же за открывающим тегом элемента `body` такую разметку:

```
<div id="main">
   <div data-win-control="WinJS.UI.HtmlControl"
     data-win-options="{uri:'/pages/header.html'}"></div>
   <h1>Print-n-Go (CH12)</h1>
   <button id="buttonPrint1">Print template #1</button>
   <button id="buttonPrint2">Print template #2</button>
   <div data-win-control="WinJS.UI.HtmlControl"
     data-win-options="{uri:'/pages/footer.html'}"></div>
</div>
```

Эта разметка предлагает одну функциональную возможность, которой не было в предыдущих упражнениях. Все дерево документа, из которого состоит пользовательский интерфейс, заключено в оболочку из элемента `div` с идентификатором

`main`. Этот внешний элемент `div` не меняет внешний вид интерфейса, но позволяет легко скрыть весь пользовательский интерфейс с помощью всего лишь одной строки кода. Теперь, если запустить приложение, перед вами должна открыться картина, показанная на рис. 12.9.

Рис. 12.9. Приложение Print-n-Go

Регистрация контракта печати

В Windows 8 основная магия, касающаяся вывода на печать, происходит на этапе инициализации приложения. Фактически, все завершается после регистрации в ходе инициализации контракта печати. Следует заметить, что инициализация приложения является всего лишь предпочтительным временем регистрации контракта печати, это можно сделать в приложении и позже. Единственной реальной причиной для этого является то, что контракт печати должен быть зарегистрирован прежде, чем пользователь сможет задействовать функциональность вывода на печать.

Как мы уже привыкли в упражнениях данной книги, метод `init` находится в основном объекте приложения. Реализация этого метода, регистрирующего контракт печати, имеет следующий вид:

```
printerApp.init = function () {
    var printManager = Windows.Graphics.Printing.PrintManager;
    var printView = printManager.getForCurrentView();

    printView.onprinttaskrequested = function (eventArgs) {
      var printTask = eventArgs.request.createPrintTask(
        "Print-n-Go", function (args) {
          args.setSource(MSApp.getHtmlPrintDocumentSource(document));
          printTask.oncompleted = onPrintTaskCompleted;
      });
    };
    function onPrintTaskCompleted(eventArgs) {
      if (eventArgs.completion ===
          Windows.Graphics.Printing.PrintTaskCompletion.failed) {
        printerApp.alert("Failed to print.");
      }
    }
}
```

Регистрация контракта печати означает, по сути, вызов метода `getForCurrentView` системного объекта `PrintManager`. Возвращаемое представление должно пройти дальнейшую настройку с помощью обработчика события заданного задания печати. Иными словами, как только пользователь через выдвигающуюся панель вызывает задание печати или как только приложение программно вызывает сервис вывода на печать, приложение должно создать задание печати.

У задания печати есть имя, по которому можно сослаться на него в принтере, и оно связано с некой работой, направленной на захват и форматирование контента для вывода на печать. Для ссылки на код вывода на печать в Windows 8 всегда используется одна строка кода. Точнее говоря, из WinJS-приложения на печать всегда выводится текущий HTML-документ, показываемый пользователю. Именно это и достигается выполнением данной строки кода:

```
args.setSource(MSApp.getHtmlPrintDocumentSource(document));
```

Далее нужно урегулировать вопрос получения методом `getHtmlPrintDocumentSource` нужного контента из объектной модели документа.

Пользовательский интерфейс вывода на печать

Давайте на время полностью отвлечемся от темы вывода контента на печать. Единственный способ вывода на печать в Windows 8 предполагает вывод диалогового окна, показанного на рис. 12.10.

Рис. 12.10. Выдвигающаяся панель при выводе на печать

После выбора принтера появится другая панель, содержащая область предварительного просмотра выводимой на печать страницы. JavaScript-разработчики не имеют возможности управлять визуализируемым контентом. Остается только отвечать за содержимое области предварительного просмотра и за вывод на печать. Панель предварительного просмотра, показанная на рис. 12.11, создается системой и не требует дополнительного кода, если не считать код, который вы уже видели.

Как показано на рис. 12.11, при намерении вывести на печать текущее содержимое экрана ничего делать не нужно — достаточно обратиться к выдвигающейся панели. Однако вполне вероятно, что вам потребуется вывести на печать только часть того контента, который в данный момент выводится на экран. Для этого нужно будет применить ряд приемов.

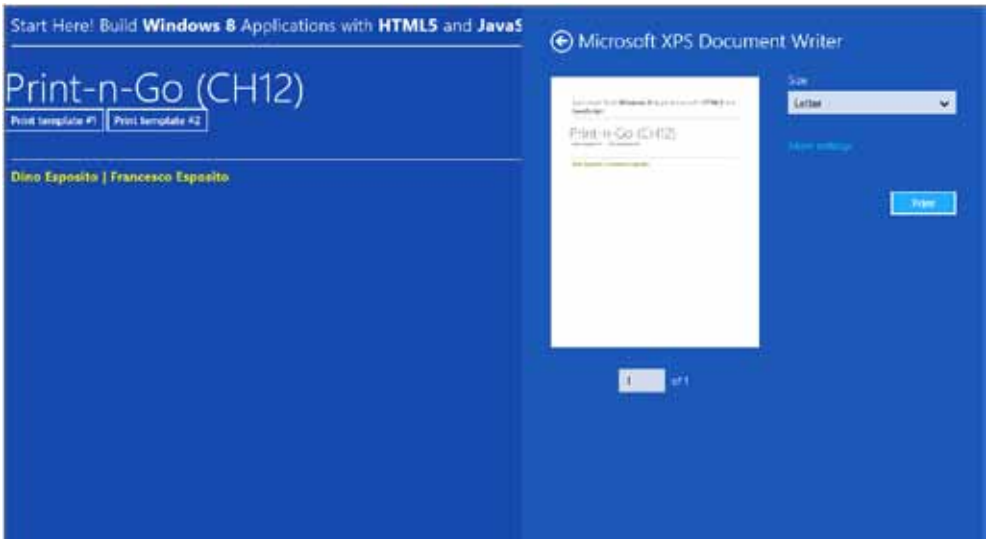

Рис. 12.11. Предварительный просмотр распечатываемой страницы

По сути, для Windows 8 и WinJS-приложения это означает, что нужно распечатать только часть HTML-контента текущего документа, представленного на экране. Вывод на печать подчиняется общим правилам браузера, то есть скрытый контент на печать не выводится, а весь видимый контент может быть дополнительно стилизован для печати.

Чтобы вывести на печать только требуемый контент, следует вооружиться таблицами стилей, предназначенными специально для вывода на печать, а также дополнительным кодом, скрывающим контент, который печатать не нужно, и оставляющим видимым лишь то, что должно попасть на принтер.

Контекстная печать контента

Вывод на печать в Windows 8 специально сделан контекстным. Пользователи в любое время ожидают предоставления возможности вывода на печать только того, что является значимым в текущем состоянии приложения. То есть в приложении, ориентированном на печать, нужно разделять контент для вывода на экран и на печать, предлагая простой способ переключения между ними.

Переключение контента в поддокументах

В файле default.html основной контент страницы уже заключен в самый внешний элемент `div`. Назначение этого внешнего элемента `div` заключается в том, чтобы вы могли разбить любой HTML-документ на два и более поддокумента: один — для обычного вывода на экран, а все остальные — для контекстной печати.

Для нашего упражнения предположим, что у вас есть два шаблона печати. Добавим две кнопки для вывода на печать двух шаблонов:

```
<button id="buttonPrint1">Print template #1</button>
<button id="buttonPrint2">Print template #2</button>
```

Определим в методе `printerApp.init` в файле printerApp.js обработчики событий щелчка на этих кнопках:

```
printerApp.init = function () {
   document.getElementById("buttonPrint1")
     .addEventListener("click", printerApp.print1, false);
   document.getElementById("buttonPrint2")
     .addEventListener("click", printerApp.print2, false);
   ...
}
```

И наконец, добавим в файл default.html одноуровневые элементы `div` для каждого поддерживаемого сценария вывода на печать. Общая структура должна выглядеть следующим образом:

```
<body>
   <div id="main">
     ...
   </div>
   <div id="print-template1">
     ...
   </div>
   <div id="print-template2">
     ...
   </div>
</body>
```

Пользовательский интерфейс помещается в элемент `div` с идентификатором `main`, а предназначенный для печати контент помещается в другие элементы `div`. Теперь нужно скрывать предназначенные для вывода на печать разделы, когда приложение запускается в нормальном режиме без выполнения печати. Точно так же нужно скрывать пользовательский интерфейс приложения и выводить на экран соответствующий контент, когда приложение должно выполнять печать.

Среда для печати

Веб-браузеры уже давно поддерживают разные среды для вывода на экран и на печать. Автор HTML-страницы может создавать ссылки на разные таблицы CSS-стилей для экрана и печати, используя атрибут `media` элементов `style` и `link`. Например, следующий код создает ссылки на файл screen.css, когда страница выводится на экран, и автоматически (без какого-либо предупреждения) переключается на файл print.css, когда страница выводится на печать:

```
<link rel="stylesheet" href="screen.css" media="screen" />
<link rel="stylesheet" href="print.css" media="print" />
```

Если атрибут `media` не указан, предполагается, что он имеет значение `all`, означающее, что одна и та же таблица стилей применяется во всех случаях.

В нашем упражнении в режиме вывода на экран нужно скрывать разделы `print-template1` и `print-template2`, а при печати страницы скрывать раздел `main`. Этого можно легко добиться с помощью атрибута `media`.

Чтобы максимально упростить задачу, мы используем элемент `style`. Этот элемент позволяет вставить стилевую информацию прямо в разметку, а не ссылаться на нее во внешнем файле:

```
<style media="print">
   #main {
     display: none;
   }
 </style>
 <style media="screen">
   #print-template1 {
     display: none;
   }
   #print-template2 {
     display: none;
   }
</style>
```

В результате два шаблона печати в режиме вывода на экран будут скрыты, а основной шаблон окажется скрытым в режиме печати.

Подготовка документа к печати

В нашем приложении Магазина Windows вам может понадобиться несколько готовых к работе шаблонов печати. Атрибут `media` не позволяет указать, какой именно шаблон печати подключен, чтобы можно было скрыть ту часть документа, которую печатать не нужно, что выглядит вполне разумно. Следовательно, нужен код, переключаемый средствами пользовательского интерфейса для программного скрытия шаблонов печати, не применимых к данному контексту. Рассмотрим, к примеру, следующую разметку в файле default.html:

```
<div id="print-template1">
   <h1 id="print-title1">Template #1</h1>
 </div>
 <div id="print-template2">
   <h1 id="print-title2">Template #2</h1>
</div>
```

Когда приложение находится в состоянии, требующем печати первого шаблона, запускается следующий код:

```
printerApp.preparePrint1 = function () {
   document.getElementById("print-template2").style.display = 'none';
   document.getElementById("print-template1").style.display = '';
```

продолжение ⇩

```
    document.getElementById("print-title1").textContent =
      "Printing template #1";
}
```

Благодаря наличию атрибута `media`, браузер автоматически выполняет код, который программно скрывает второй шаблон при намерении вывести на печать первый. В то же время шаблон заполняется или обновляется свежим контентом, чтобы лучше отражать состояние приложения. В этот момент шаблон печати остается единственной видимой частью документа, следовательно, единственной частью, которая будет распечатана.

В общем и целом, для управления событиями щелчка на кнопках печати в вашем приложении нужен следующий код:

```
printerApp.print1 = function () {
    printerApp.preparePrint1();
    Windows.Graphics.Printing.PrintManager.showPrintUIAsync();
 }
 printerApp.print2 = function () {
    printerApp.preparePrint2();
    Windows.Graphics.Printing.PrintManager.showPrintUIAsync();
}
```

Методы `preparePrint1` и `preparePrint2` зеркальны по отношению друг к другу; с их помощью происходит просто добавление распечатываемых данных к поддереву документа и скрытие ненужных шаблонов печати. Оба этих метода определяются в файле printerApp.js. Методы `print1` и `print2` вызываются из обработчиков щелчков на кнопках печати в пользовательском интерфейсе.

Системный метод `showPrintUIAsync` отвечает за программный вывод на экран выдвигающейся панели печати, той самой панели, которую пользователь может вызвать, коснувшись правого края экрана. На рис. 12.12 показаны выводимые на печать данные в режиме предварительного просмотра, когда на принтеры отправляется второй шаблон.

В общем, вывод на печать из WinJS-приложения сводится к редактированию текущей страницы, чтобы на браузер отправлялся только тот контент, который требуется распечатать.

ПРИМЕЧАНИЕ

Поскольку WinJS-приложения по своей сути требуют применения браузера, можно решить, что вывод на печать можно выполнять привычным способом, практиковавшимся в JavaScript: вызывая метод window.print. Хотя такой подход работает, он применим только в самых простых ситуациях. Метод window.print рекомендуется применять, только если имеется всего один шаблон печати. После вызова метода window.print вы уже не сможете ни управлять процессом печати, ни обновлять и настраивать печатаемый контент.

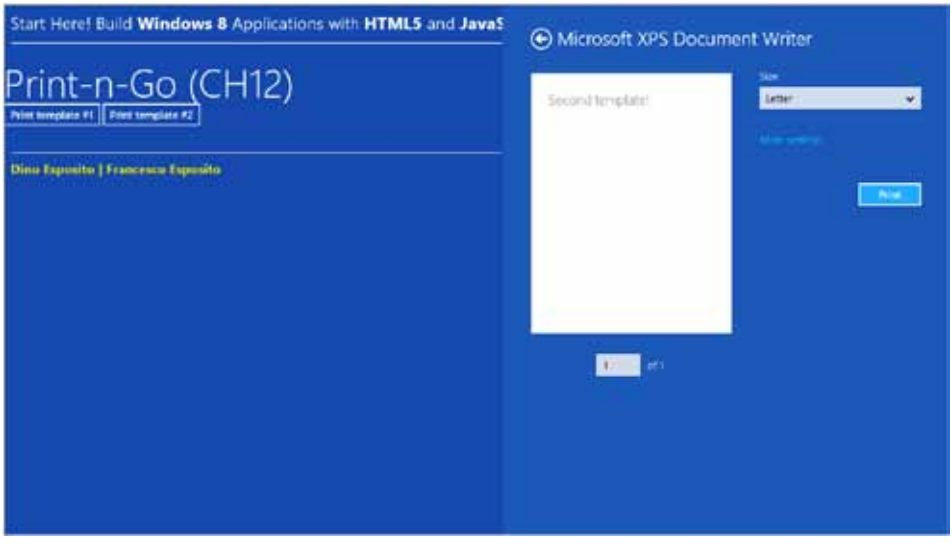

Рис. 12.12. Управление печатаемым контентом

Работа с GPS

В завершение данной главы мы выясним, как получить информацию о текущем местоположении пользователя. Точное (или даже приблизительное) местоположение пользователя является очень ценной информацией, поскольку позволяет разрабатывать специальные службы для пользователей.

Определение широты и долготы

У устройств с Windows 8, оснащенных системой GPS, могут запрашиваться основные данные о местоположении, такие как широта и долгота. Доступ к устройству «упакован» в красивый и удобный прикладной программный интерфейс.

Предварительная настройка проекта

Сначала нужно создать новый проект и выполнить все шаги, которые выполнялись в предыдущих приложениях. Пользовательский интерфейс может быть максимально простым, как тот, который показан на рис. 12.13: одна кнопка и пара текстовых полей для вывода широты и долготы.

Рис. 12.13. Основной пользовательский интерфейс учебного приложения геолокации

Наиболее важным шагом в предварительной настройке проекта является подключение возможности определять местоположение в манифесте приложения. Откройте в проекте файл манифеста, перейдите на вкладку Capabilities (Возможности) и установите флажок Location (Местоположение), как показано на рис. 12.14.

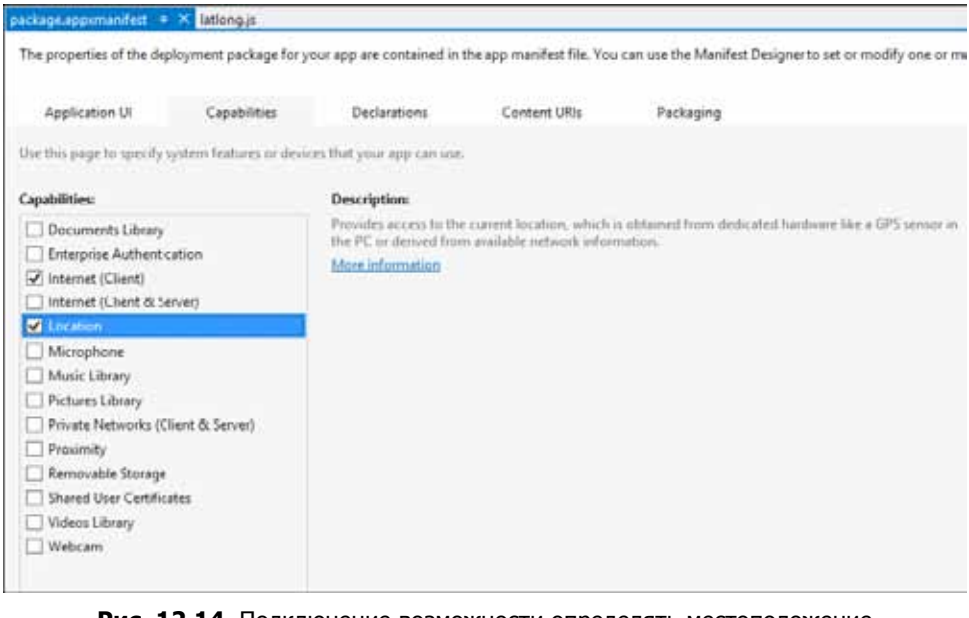

Рис. 12.14. Подключение возможности определять местоположение

Приложение получает информацию о местоположении от GPS-датчика, если этот датчик доступен на устройстве. Если нет, компонент выводит информацию о местоположении пользователя на основе IP-адреса устройства, с которым оно подключено к Интернету.

Чтение географического положения

Далее показан код, необходимый для отправки запроса к подсистеме определения местоположения. Этот код можно скопировать непосредственно в JavaScript-файл latlong.js, относящийся к основному приложению:

```
var latLongApp = latLongApp || {};
 var geolocator = null;

 latLongApp.init = function () {
   document.getElementById("buttonLocation")
     .addEventListener("click", latLongApp.getPosition);
 }

 latLongApp.getPosition = function () {
   geolocator = new Windows.Devices.Geolocation.Geolocator();
   var position = geolocator.getGeopositionAsync().then(latLongApp.display,
     latLongApp.error);
}
```

После получения ссылки на системный компонент геолокации нужно просто выполнить запрос методом `getGeopositionAsync`. Как обычно, метод работает в асинхронном режиме и устанавливает объект обязательства, получающий информацию о широте и долготе. Пользователь получает необработанную информацию методом `latLongApp.display`, код которого имеет следующий вид:

```
latLongApp.display = function (location) {
   document.getElementById("lat").innerHTML = location.coordinate.latitude;
   document.getElementById("long").innerHTML = location.coordinate.longitude;
}
```

Конечный результат показан на рис. 12.15.

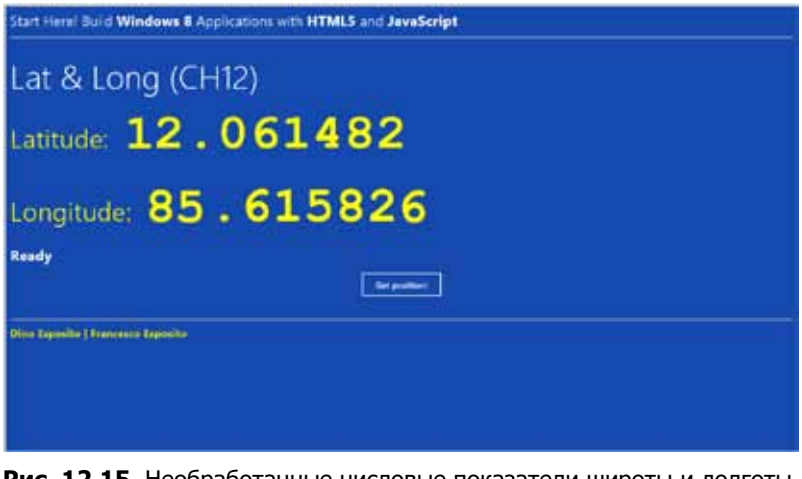

Рис. 12.15. Необработанные числовые показатели широты и долготы, предлагаемые пользователю

Следует заметить, что пользователь в любое время может отозвать разрешение на работу с диспетчером определения местоположения, перейдя на страницу Settings (Настройки) и отключив соответствующий режим на странице Permissions (Права доступа), как показано на рис. 12.16. Обычно при первом запуске приложения система просит пользователя явным образом подключить эту возможность. Следовательно, решение о том, разрешить или запретить, пользователь принимает сам. Та же логика имела место и в рассмотренном ранее примере с веб-камерой.

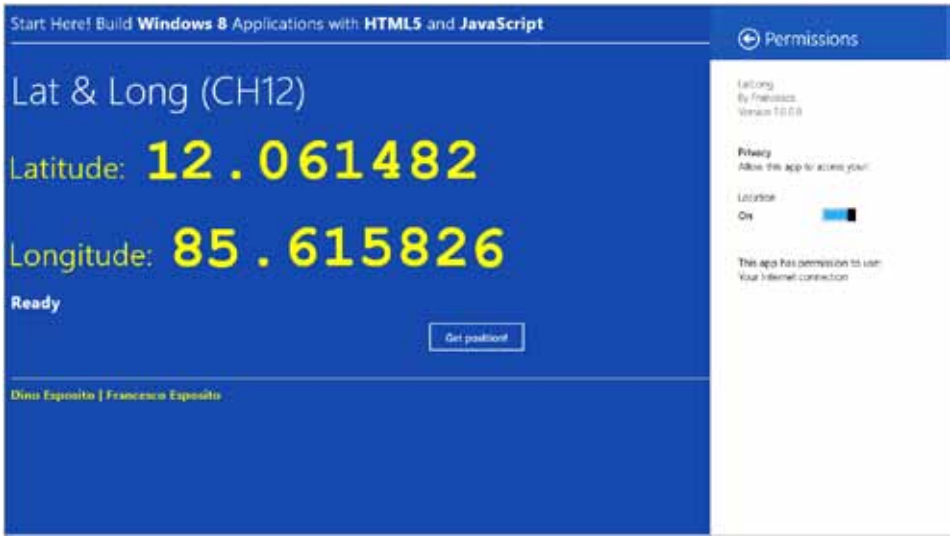

Рис. 12.16. Страница настройки, на которой пользователь может разрешить или запретить приложению определять местоположение

Использование данных геолокации

Данные геолокации являются весьма деликатной информацией, поскольку благодаря ей потенциально весь мир может узнавать, где находится тот или иной человек в то или иное время. И хотя местонахождение — это, конечно же, вопрос частной жизни, это также и вопрос маркетинга. По этим двум причинам Windows 8, как и другие мобильные операционные системы и веб-браузеры, оставляет решение о доступности данных о местоположении за пользователем.

Как разработчику прикладной программный интерфейс геолокации предоставляет вам последовательности чисел, описывающих широту и долготу. Хотя они уникальном образом указывают на конкретное местоположение на карте мира, их польза для приложений весьма сомнительна. Чтобы они стали действительно полезными, необработанные числа обычно нужно преобразовывать в более понятные человеку географические данные, такие как город и страна.

Подключение к приложению библиотеки Bing SDK

Согласно документации по Windows 8, объект, который вы получаете в результате вызова метода `getGeopositionAsync`, должен содержать свойство с типом данных `CivicAddress` (городской адрес). Как следует из названия, от этого типа данных ожидается выдача точной информации об улице, городе и стране, соответствующей значениям широты и долготы. Но если полностью и внимательно прочитать документацию, то обнаружится, что Windows 8 не устанавливает модуль, способный отобразить геопозицию на городской адрес. Это означает, что следующий код, не выбросив никакого исключения, откажется выдавать какую-либо значимую информацию:

```
document.getElementById("address").innerHTML =
  location.civicAddress.country;
```

Оказывается, для отображения широты и долготы на городской адрес (если таковой существует) нужно из проекта сослаться на библиотеку Microsoft Bing SDK. Ссылка на эту библиотеку также пригодится для визуализации местоположения пользователя на карте. Но сначала нужно загрузить Bing SDK с URL-адреса http://bit.ly/N9NFQN. Затем для ссылки на Bing SDK в вашем проекте щелкните правой кнопкой мыши на узле References (Ссылки) и выберите пункт Add Reference (Добавить ссылку). После этого установите флажок Bing Maps for JavaScript (Bing Maps для JavaScript), как показано на рис. 12.17.

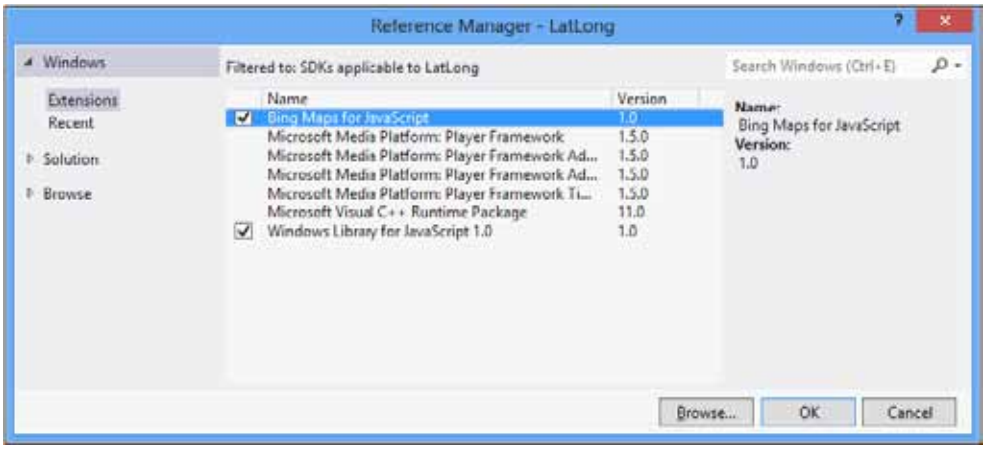

Рис. 12.17. Ссылка на Bing SDK в приложении

Перед тем как приступить к работе с библиотекой Bing SDK, нужно обратить внимание еще на несколько вещей. В первую очередь, со своих страниц требуется сослаться на JavaScript-файл для Bing SDK. Добавьте в файл default.html следующий тег `script`:

```
<!-- Ссылки на Bing Map Control -->
 <script type="text/javascript"
  src="ms-appx:///Bing.Maps.JavaScript//js/veapicore.js"></script>
```

Для работы Bing SDK требует учетные записи. Учетные записи позволяют библиотеке обеспечить отслеживание пользователей и функций. Учетные записи для Bing состоят из ключа, создаваемого после регистрации на Bing-портале по адресу http://www.bingmapsportal.com. На рис. 12.18 показана страница, на которую вы попадете после успешной регистрации и создания своего ключа для приложения Магазина Windows.

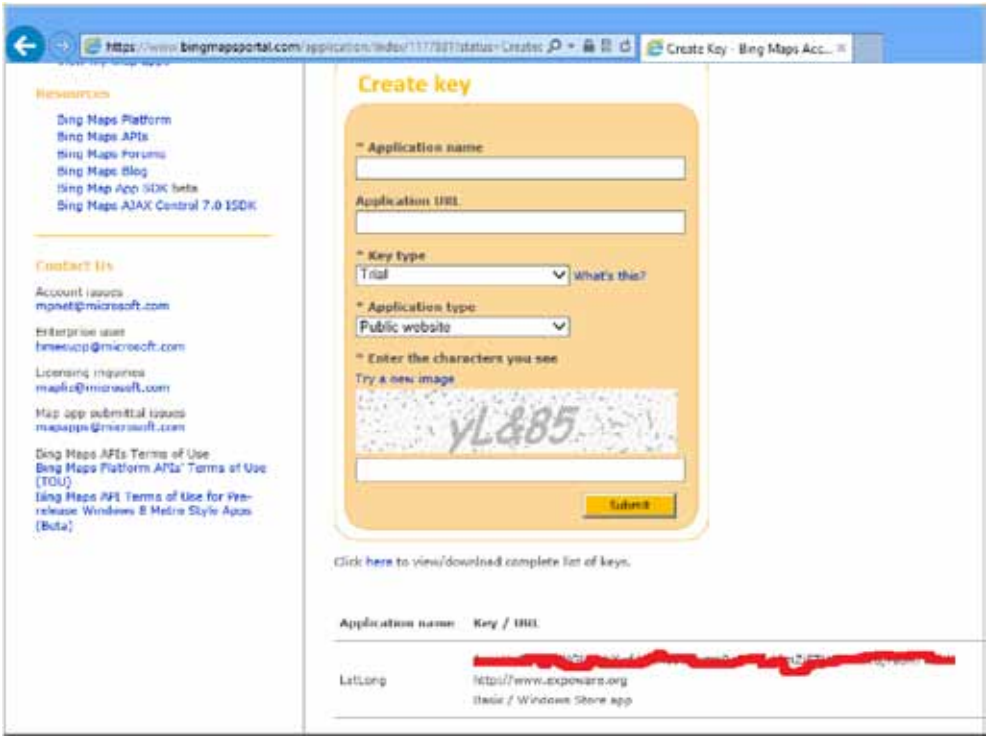

Рис. 12.18. Получение ключа для работы с библиотекой Bing SDK

Обязательно запишите где-нибудь ключ, потому что он понадобится для использования Bing SDK в ваших приложениях:

```
latLongApp.BingKey = "...";
```

Перед использованием библиотеку Bing нужно инициализировать. Это делается в методе `init` приложения. Добавьте в метод `atLongApp.init` следующий код:

```
latLongApp.init = function () {
    ...
    Microsoft.Maps.loadModule('Microsoft.Maps.Map', { culture: 'en-us' });
}
```

Теперь все готово для вывода на экран карты и извлечения адресной информации.

Показ местоположения на карте

Приложения Магазина Windows, созданные средствами JavaScript, для показа карты не могут применять какие-либо встроенные компоненты. Тем не менее в страницу можно включить элемент `div` и использовать библиотеку Bing SDK для вывода карты в границах этого элемента `div`. Добавьте в файл default.html там, где вам понравится например, перед нижним колонтитулом, следующий код:

```
<div id="map" style="margin:2px;width:500px;height:400px; border:solid 2px #0ff;">
</div>
```

Точно указать размер элемента `div` можно, присвоив значения его атрибутам `width` и `height`. Указание точных размеров не позволит карте занять весь экран. В файле latLongApp.js нужно создать несколько новых функций. Добавьте следующую функцию, которая подготовит почву для вывода карты:

```
latLongApp.setupMap = function () {
   try {
     var mapOptions = {
       credentials: latLongApp.BingKey,
       mapTypeId: Microsoft.Maps.MapTypeId.road,
       width: 500,
       height: 400
     };
     var mapDiv = document.getElementById("map");
     latLongApp.map = new Microsoft.Maps.Map(mapDiv, mapOptions);
   }
   catch (e) {
     latLongApp.alert(e.message);
   }
};
```

Важно, чтобы значения свойств `width` и `height` совпадали с размерами элемента `div`, который планируется использовать для визуализации карты. Эта новая функция вызывается из метода обратного вызова, который запускается при получении координат. Метод `latLongApp.display` выглядит так:

```
latLongApp.display = function (location) {
   document.getElementById("lat").innerHTML = location.coordinate.latitude;
   document.getElementById("long").innerHTML = location.coordinate.longitude;

   // Очистка элемента div и настройка объекта карты
   document.getElementById("map").innerHTML = "";
   latLongApp.setupMap();

   // Ссылка на центр карты и установка ее на текущее местоположение
   var mapCenter = latLongApp.map.getCenter();
   mapCenter.latitude = location.coordinate.latitude;
   mapCenter.longitude = location.coordinate.longitude;

   // Настройка представления карты
   latLongApp.map.setView({ center: mapCenter, zoom: 16 });
}
```

Полученный таким способом результат уже полезен, но его можно еще немного улучшить, добавив так называемую канцелярскую кнопку, точнее идентифицирующую местоположение пользователя. Эта кнопка добавляется следующим образом:

```
latLongApp.addPushPin = function (location) {
    latLongApp.map.entities.clear();
    var pushpin = new Microsoft.Maps.Pushpin(location, null);
    latLongApp.map.entities.push(pushpin);
}
```

Теперь вызовите в конце метода `latLongApp.display` метод `addPushPin`:

```
latLongApp.addPushPin(mapCenter);
```

Результат показан на рис. 12.19.

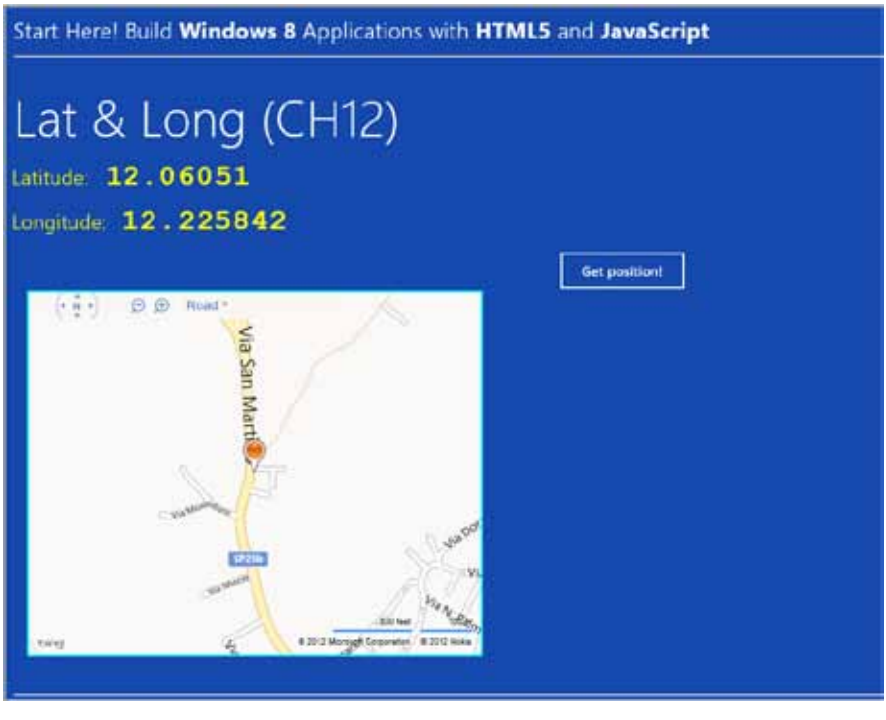

Рис. 12.19. Текущее местоположение на карте — кнопка указывает на точную позицию с заданными координатами

Получение информации об адресе

Завершающий шаг упражнения состоит в отображении координат на более полезную информацию, такую как названия улицы, города и страны. Как уже упоминалось,

соответствующий код встроен в Windows 8. Однако непосредственно обратившись к службе Bing, можно получить JSON-строку, содержащую все детали.

Добавьте в файл latLongApp.js новую функцию. Эта функция выполняет удаленный вызов в службе Bing, передавая ей координаты и получая в ответ информацию о точном городском адресе. URL-адрес вызова имеет следующий формат:

```
http://dev.virtualearth.net/REST/v1/Locations/lat,long?o=json&key=...
```

Для отправки вызова по URL-адресу используется объект `WinJS.xhr`, с которым мы впервые встретились в главе 11. Код, превращающий координаты в адрес, выглядит так:

```
latLongApp.convertToAddress = function (location) {
   var url = "http://dev.virtualearth.net/REST/v1/Locations/" +
       location.latitude +
       "," +
       location.longitude +
       "?o=json&key=" +
       latLongApp.BingKey;

   // Вызов службы Bing
   WinJS.xhr({ url: url }).then(function (response) {
     var data = JSON.parse(response.responseText);
     var address = data.resourceSets[0].resources[0].name;

     // Подготовка информационного блока к добавлению карты
     var infoboxOptions = { zIndex: 3, title: address };
     var defaultInfobox = new Microsoft.Maps.Infobox(
       location, infoboxOptions);
     latLongApp.map.entities.push(defaultInfobox);
   });
}
```

Служба Bing возвращает сложную JSON-строку, которая после ее синтаксического разбора превращается в JavaScript-объект, содержащий полный адрес в свойстве `name`:

```
var address = data.resourceSets[0].resources[0].name;
```

Вывести адрес на экран можно несколькими способами. Например, можно визуализировать его на странице в виде простого текста. Можно также создать информационный блок и показывать его на карте рядом с установленной канцелярской кнопкой.

Для вызова службы Bing нужны только широта и долгота, на которых находится пользователь (плюс, конечно же, ваш Bing-ключ), поэтому функцию `convertToAddress` можно вызвать в любое время. Однако в учебном приложении этот вызов лучше поместить сразу же после вызова функции, добавляющей канцелярскую кнопку:

```
latLongApp.display = function (location) {
   ...
   latLongApp.addPushPin(mapCenter);
   latLongApp.convertToAddress(mapCenter);
}
```

Конечный результат показан на рис. 12.20.

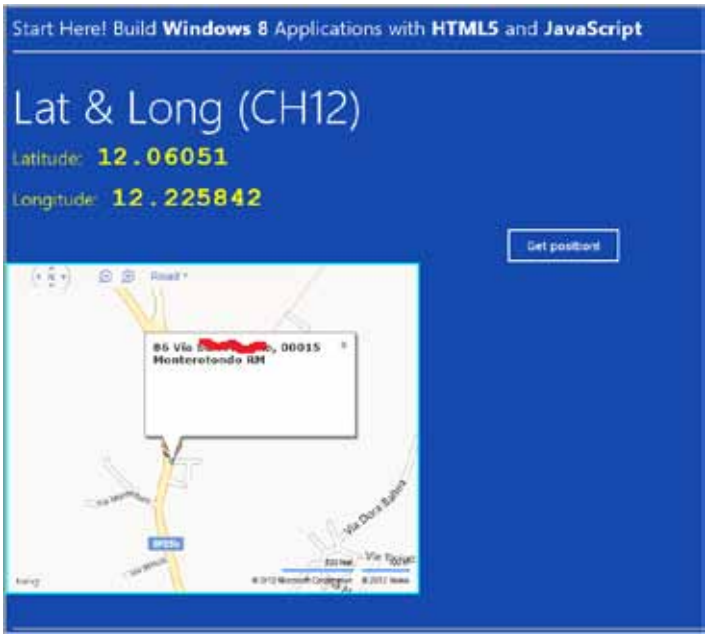

Рис. 12.20. Демонстрация информационного блока с городским адресом, соответствующим заданным координатам

Выводы

Как настольные компьютеры, так и мобильные устройства (то есть ноутбуки, планшеты и смартфоны) в избытке оснащены датчиками и специализированными устройствами, которые позволяют разработчикам создавать более сложные приложения, открывающие для пользователей уникальные перспективы. Возможность с высокой точностью определять текущее местоположение устройства позволяет персонализировать основной контент и пользовательский интерфейс приложения. Другие датчики, в том числе не упомянутые в этой главе, могут помочь приложению определить, как устройство используется, двигается или поворачивается.

Программирование приложений Магазина Windows требует от разработчика знакомства с широким спектром датчиков, играющих роль шлюза между окружающим миром и устройством. В этой главе мы узнали, как работать с веб-камерой, принтером и GPS. Но важнее то, что мы исследовали методы, которые вполне применимы и к другим датчикам, например к спидометру или гироскопу.

В следующей главе мы займемся самым характерным элементом устройств на базе Windows — живыми плитками.

13 Живые плитки

Есть ученые-садисты, которые бросаются искать ошибки, вместо того чтобы устанавливать истину.

Мария Кюри

Пользовательские интерфейсы Microsoft Windows 8 и Windows Phone характеризуются красочными блоками, которые напоминают многим пользователям те старомодные значки, которые сделали ранние версии Microsoft Windows столь популярными. Однако новые блоки значительно больше значков и под управлением операционной системы выводятся на экран вплотную друг к другу. Эти блоки называются *плитками* (tiles). Термин «плитки» относится главным образом к форме и размеру графического элемента. Плитки в Windows 8 (как и в Windows Phone) имеют дополнительную очень интересную способность: они могут выводить на экран специализированную информацию, генерируемую приложением в соответствии с пожеланиями пользователя, установившего приложение. Такие плитки называются *живыми* (live tiles).

Значок — это статичное изображение, позволяющее пользователю быстро идентифицировать приложение. Но он никогда не меняется, чтобы отразить своим видом текущее состояние приложения. В отличие от значка, живая плитка является своеобразным дополнением приложения, передающим некий контент операционной системе, которая затем выводит на экран содержащуюся в нем информацию пользователю, даже когда приложение отключено от сети или вообще не запущено.

С точки зрения разработчика, программирование живых плиток требует знакомства с новым прикладным программным интерфейсом и таким понятием, как *уведомление приложением* (application notification). В данной главе мы выполним

упражнение, в ходе которого добавим живые плитки к приложению TodoList, созданному в предыдущих главах.

Что собой представляет живая плитка

На рис. 13.1 показан начальный экран машины под управлением Windows 8. Каждый блок пользовательского интерфейса представляет собой установленное приложение. Хотя живая плитка может быть активной и ее информация может поддерживаться в актуальном состоянии, во многих случаях (например, когда приложение не подключено к сети) она представляет собой новую и более привлекательную версию обычного значка, знакомого всем по предыдущим версиям Windows. Все показанные плитки находятся в статичном состоянии, и каждая выводит на экран только логотип и имя приложения.

Рис. 13.1. Плитки на начальном экране машины под управлением Windows 8

Плитки в действии

Пользователи могут оперировать плитками практически так же, как значками в более ранних версиях Windows. То есть пользователь может перемещать плитки, группировать их, делать крупнее или мельче, а также включать и выключать режим

оперативных уведомлений. В конечном счете плитки представляют собой обновленные значки, обогащенные возможностью получения оперативных уведомлений от базовых приложений.

Перемещение живых плиток

Windows 8 создает новую плитку для каждого установленного приложения и закрепляет ее на начальном экране. После этого пользователь может убрать плитку с начального экрана, если не хочет ее там видеть, или даже вообще удалить вместе с самим приложением. На рис. 13.2 показано контекстное меню, которое появляется, когда пользователь щелкает на плитке правой кнопкой мыши.

Рис. 13.2. Контекстное меню плитки

Как видно на рисунке, доступными являются команды удаления приложения с начального экрана, полного удаления приложения, изменения размеров плитки, а также включения и выключения режима оперативных уведомлений от приложения.

Точно так же, как и при работе со значками в Windows 7 и более ранних версиях Windows, пользователи могут перемещать плитки и составлять из них горизонтальный прокручиваемый список. Но, в отличие от значков, плитки Windows 8 нельзя группировать в папки. Как показано на рис. 13.3, перемещение плиток осуществляется предельно просто, пользователю достаточно перетащить плитку мышью (или пальцем в случае сенсорного экрана).

Рис. 13.3. Перемещение плитки

Изменение размеров плиток

Плитки могут быть большие или малые. Малая плитка представляет собой прямоугольный блок размером 130 × 130 пикселов, большая плитка примерно в два раза шире. Как видно на рис. 13.3, перемещаемая плитка приложения Microsoft Internet Explorer является малой, а плитка приложения Mail в левом верхнем углу — большой.

Плитки для нестандартных приложений всегда создаются малыми. Только пользователь может изменить размер плитки, обратившись к контекстному меню. Из-за возможности смены размеров плитки создают сложности тем разработчикам, кто хотел бы встроить в свои приложения механизм уведомлений. Чтобы решить проблему, нужно применять разные графические шаблоны для малых и больших плиток, как мы сделаем в нашем упражнении.

Приложения без плиток

В контекстном меню плитки (см. рис. 13.2) есть команда Unpin From Start (Убрать с начального экрана), которая дает пользователю возможность убрать плитку приложения с начального экрана. Следует иметь в виду, что удаление плитки не означает удаления приложения. Это просто означает, что приложение убирается из списка приложений, видимых на начальном экране. В отличие от этой команды, команда Uninstall (Удалить) полностью удаляет приложение с устройства.

Чтобы запустить приложение, не имеющее плитки на начальном экране, нужно просто коснуться нижней части экрана (или щелкнуть правой кнопкой мыши за пределами плиток), вызвав контекстное меню с командой All Apps (Все приложения), предоставляющей доступ к полному списку установленных приложений, в том числе приложений без плиток (рис. 13.4). Можно также начать набирать имя приложения на начальном экране, и тогда пользовательский интерфейс оставит на экране только те приложения, чьи имена соответствуют набранным символам.

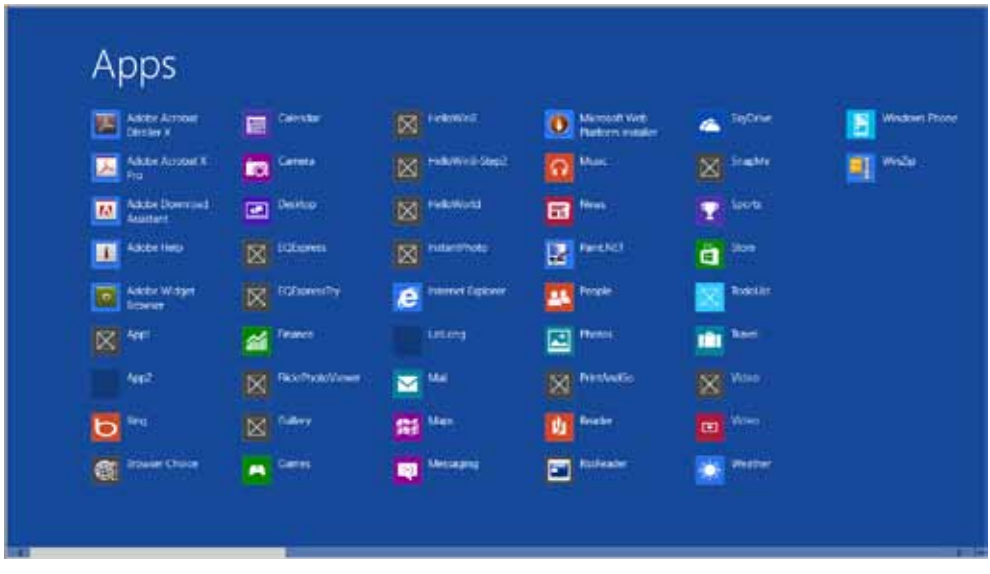

Рис. 13.4. Список всех установленных приложений

Создание живых плиток для базового приложения

Наиболее интересной стороной работы с плитками является их оживление путем программирования уведомлений. Чтобы решить задачу оживления плиток, создадим сначала новое учебное приложение, выводящее на плитке какой-нибудь статический текст. Затем мы доработаем приложение, создав более сложный пример, в котором выводимый текст будет отражать данные и состояние самого приложения.

Подготовка приложения

Создайте новое приложение Магазина Windows, используя шаблон Blank App (Пустое приложение), и добавьте в папку Pages обычные файлы header.html и footer.html. Потребуется также включить в файл default.css несколько новых стилей и добавить JavaScript-файл, названный по имени приложения tilesDemoApp.js. Кроме того, нужно включить в файл default.js следующую строку кода начальной загрузки:

```
app.onready = function (args) {
   tilesDemoApp.init();
};
```

Исходный контент файла tilesDemoApp.js выглядит так:

```
var tilesDemoApp = tilesDemoApp || {};
 tilesDemoApp.init = function () {
   // Сюда помещается остальной код
};
```

Код страницы в файле default.html должен выглядеть следующим образом:

```
<!DOCTYPE html>
 <html>
 <head>
   <meta charset="utf-8" />
   <title>Simple Tiles Demo</title>

   <!-- Ссылки WinJS -->
 <link href="//Microsoft.WinJS.1.0/css/ui-dark.css" rel="stylesheet" />
 <script src="//Microsoft.WinJS.1.0/js/base.js"></script>
 <script src="//Microsoft.WinJS.1.0/js/ui.js"></script>

   <!-- Ссылки TilesDemo -->
 <link href="/css/default.css" rel="stylesheet" />
 <script src="/js/default.js"></script>
 <script src="/js/tilesdemoapp.js"></script>
 </head>
 <body>
   <div data-win-control="WinJS.UI.HtmlControl"
     data-win-options="{uri:'/pages/header.html'}"></div>

   <h1>Simple Tiles Demo (CH13)</h1>
   <div data-win-control="WinJS.UI.HtmlControl"
     data-win-options="{uri:'/pages/footer.html'}"></div>
 </body>
</html>
```

Пока все хорошо, но еще ничего в приложении не вдохнуло жизнь в статичную по умолчанию плитку.

Объект уведомлений

Для оживления плиток нужно создать уведомление. В данном контексте уведомление является экземпляром объекта `Windows.UI.Notifications.TileNotification`. При создании экземпляра объекта уведомлений нужно передать ему какие-нибудь данные, идентифицирующие макет, который вы наметили для контента плитки. В течение жизненного цикла приложения объект уведомлений создается только один раз. Но впоследствии контент плитки можно менять столько раз, сколько требует логика работы приложения.

Объект уведомлений действует в качестве своеобразного моста между приложением и системой. После создания уведомление остается на своем месте некоторое время, даже если приложение не работает или отключено от сети.

Создание уведомления в приложении

Для добавления к приложению живой плитки нужно пройти три этапа. Сначала выбирается макет для текста плитки. Затем к макету добавляются данные приложения. И наконец, с помощью шаблона создается объект уведомлений и добавляется к системному списку.

С Windows 8 поставляется множество шаблонов плиток. Их можно найти в перечислении `Windows.UI.Notifications.TileTemplateType`. Каждый член перечисления ссылается на свой макет с несколькими местами, выделенными для заполнения текстом и (или) изображениями. Чаще всего используется следующий шаблон плитки:

```
Windows.UI.Notifications.TileTemplateType.tileSquareText02
```

Шаблон состоит из двух строк текста, автоматически стилизованных под заголовок и подзаголовок какого-нибудь приложения. Первая строка текста располагается в верхней части плитки и выводится крупным шрифтом. Вторая строка располагается в ее нижней части и выводится шрифтом меньшего размера.

В действительности шаблон является XML-строкой, но вам как разработчику не стоит слишком сильно вдаваться в детали XML. Достаточно получить содержимое шаблона и поработать над ним, заменив ряд элементов. К коду запуска вашего приложения нужно добавить следующий код в файле tilesDemoApp.js:

```
tilesDemoApp.init = function () {
    var template = Windows.UI.Notifications.TileTemplateType.tileSquareText02;
    var xml = Windows.UI.Notifications.TileUpdateManager.
      getTemplateContent(template);

    // Сюда помещается остальной код
}
```

Возвращаемый XML-контент зависит от выбранного шаблона. Что касается нашего шаблона, он состоит из двух элементов `text`, которые нужно заполнить текстом, выводимым на плитке. Для добавления контента, характерного для приложения, введите следующий код:

```
tilesDemoApp.init = function () {
    var template = Windows.UI.Notifications.TileTemplateType.tileSquareText02;
    var xml = Windows.UI.Notifications.TileUpdateManager.
    getTemplateContent(template);
    var textElements = xml.getElementsByTagName("text");

    // Заполнение текстом выделенных мест в шаблоне плитки
    textElements[0].innerText = "Title";
    textElements[1].innerText = "This is the subtitle";

    // Сюда помещается остальной код
}
```

И наконец, нужно зарегистрировать плитку в операционной системе, чтобы ее контент правильно визуализировался в начальном экране. Для этого в функцию `init` в файле tilesDemoApp.js нужно добавить несколько строк кода. В результате функция `init` должна приобрести окончательный вид:

```
tilesDemoApp.init = function () {
    var template = Windows.UI.Notifications.TileTemplateType.tileSquareText02;
    var xml = Windows.UI.Notifications.TileUpdateManager.
    getTemplateContent(template);
    var textElements = xml.getElementsByTagName("text");

    // Заполнение текстом выделенных мест в шаблоне плитки
    textElements[0].innerText = "Title";
    textElements[1].innerText = "This is the subtitle";

    // Создание и регистрация объекта уведомления
    var liveTile = new Windows.UI.Notifications.TileNotification(xml);
    Windows.UI
      .Notifications
      .TileUpdateManager.createTileUpdaterForApplication().update(liveTile);
}
```

Сначала из XML-шаблона создается новый объект уведомлений, а затем добавляется к системному списку живых плиток для установленных приложений. Для каждой активной в данный момент живой плитки система поддерживает объект, отвечающий за периодический вывод самого последнего контента на начальном экране. На рис. 13.5 показана живая плитка учебного приложения, запущенного на машине под управлением Windows 8.

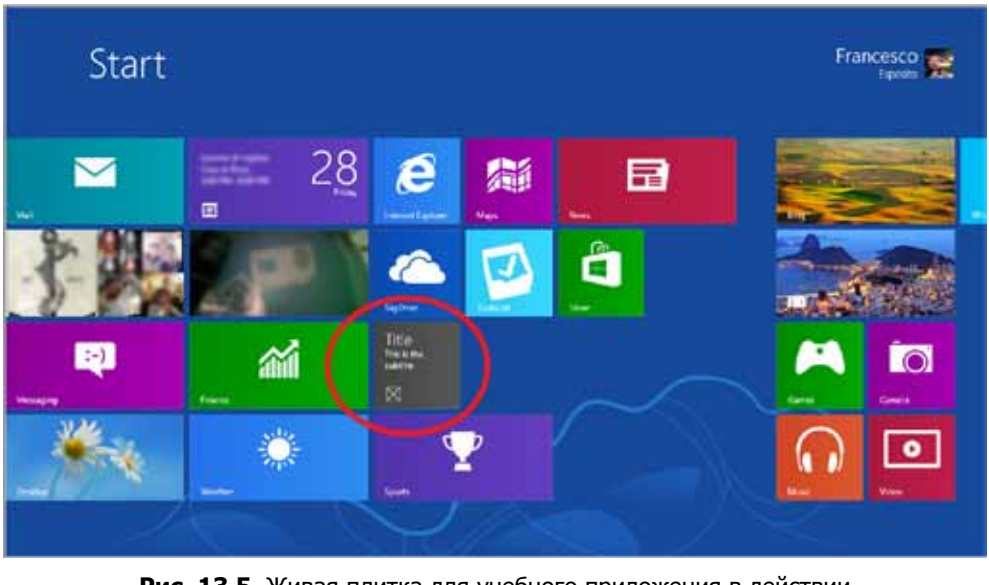

Рис. 13.5. Живая плитка для учебного приложения в действии

Независимо от выбранного шаблона, исходный логотип приложения всегда располагается в нижней части плитки. Остальная часть плитки находится в вашем распоряжении и может быть заполнена в соответствии с шаблоном.

Добавление живых плиток к существующему приложению

В конечном счете добавление живой плитки к приложению — не такая уж сложная задача. Все, что для этого требуется, сводится к нескольким вызовам предоставляемого системой прикладного программного интерфейса. Трудности возникнут, если вы попытаетесь добавить живые плитки к реально существующему приложению. В этом случае самой трудной частью работы окажется принятие решения о том, какие данные должны попасть на живую плитку, каким образом они должны извлекаться, как часто они будут обновляться и, разумеется, какой из шаблонов больше всего подойдет. Кроме того, может потребоваться поддержка как больших, так и малых плиток, чтобы у ваших пользователей создалось целостное впечатление от работы с Windows 8.

Возвращение к приложению TodoList

В главе 10 мы выполнили упражнение, в котором создали версию приложения TodoList, обеспечивающую сохранение данных. Окончательная версия приложения позволяла создавать, редактировать и удалять задания, и каждое задание сохранялась в роуминговой папке, обеспечивая тем самым попадание параметров приложения в облако и их совместное использование с другой копией того же приложения, установленной на другом персональном компьютере или устройстве.

В данном упражнении мы усовершенствуем приложение TodoList, созданное в главе 10, включив в него живые плитки.

Подготовка почвы

Сделайте копию проекта из главы 10 и назовите ее TodoList-Local. Возможно, вам захочется переименовать файлы проекта, чтобы в них отражался номер текущей главы. Возможно, вам также захочется откорректировать контент страницы в файле default.html из чисто графических соображений. Откройте файл default.html в текстовом редакторе в Microsoft Visual Studio и отредактируйте следующую строку:

```
<h1> TO-DO List (CH13) </h1>
```

Теперь все должно быть готово, чтобы вспомнить об устройстве данного приложения и составить планы относительно использования живых плиток. На рис. 13.6

показан основной пользовательский интерфейс приложения, который практически такой же, как был в главе 10. А на рис. 13.7 показана плитка, которую Windows 8 по умолчанию создает для приложения.

Рис. 13.6. Базовое приложение, дополняемое живыми плитками

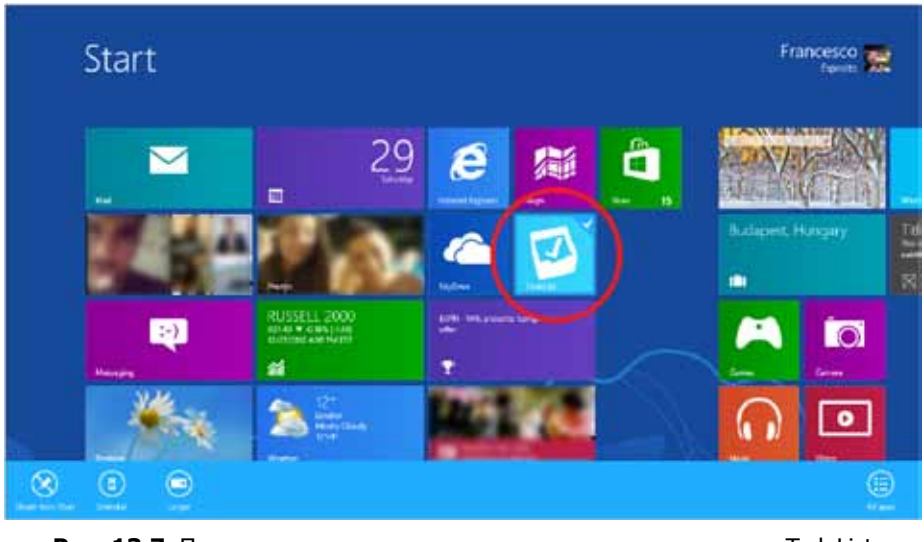

Рис. 13.7. Предлагаемая по умолчанию плитка для приложения TodoList

Когда вы в качестве пользователя щелкните правой кнопкой мыши на плитке, появится контекстное меню с доступными командами. Если живые плитки не активны,

соответствующие команды пользователю не доступны. Windows 8 определяет, поддерживает ли приложение живые плитки, и добавляет в контекстное меню еще одну команду, включающую и выключающую режим оперативного обновления.

Определение редактируемых файлов проекта

В предыдущем элементарном упражнении, как вы, наверное, заметили, добавление к приложению живых плиток требует совсем немного кода. Единственным файлом проекта, на который следует обратить внимание, является todolist.js. Фактически, живые плитки не оказывают влияния ни на общий пользовательский интерфейс, ни на манифест приложения, также они не требуют специальных разрешений.

Поскольку код управления живыми плитками работает изолированно, его нетрудно убрать в отдельный файл, который затем будет вызываться из файла основного сценария приложения. Поэтому далее при выполнении упражнения мы займемся созданием нового JavaScript-файла, а также, что, возможно, важнее, выбором более подходящего шаблона и контента, предназначенного для вывода на экран.

Реализация живых плиток

Живые плитки были задуманы как средство, дающее возможность разработчикам представить полезную информацию пользователям, не заставляя их даже открывать приложение. Живые плитки служат напоминанием о функциональности или контенте приложения, а также способствуют более частому обращению к приложению. Для успешного применения живых плиток они должны своевременно и привлекательно предоставлять полезную информацию, связанную с приложением. Поэтому выбор шаблона плитки и выводимых на ней данных является ключевым.

Подготовка почвы для живых плиток

Добавьте в проект новый JavaScript-файл. Назовем его liveTiles.js. Для начала нужно в только что созданный файл добавить следующий код:

```
var liveTilesManager = liveTilesManager || {};
 liveTilesManager.enable = function (listOfTasks) {
   // Сюда помещается остальной код
}
```

Кроме того, на файл liveTiles.js нужно сослаться из файла default.html. Поэтому откройте в редакторе Visual Studio файл default.html и добавьте к нему такую строку:

```
<script src="/js/livetiles.js"></script>
```

Теперь все готово к включению в диспетчер живых плиток более сложного кода.

Выбор шаблонов плиток

Windows 8 поставляется с большим перечнем готовых шаблонов для плиток. Здесь можно найти шаблоны как для больших, так и для малых плиток. По сути, шаблон плитки является небольшим фрагментом XML-данных, содержащим выводимую на плитке информацию. Обычно живая плитка состоит из изображений и одной или нескольких строк текста. Дополнительные сведения о доступных шаблонах плиток можно получить, обратившись по URL-адресу http://msdn.microsoft.com/en-us/library/windows/apps/hh761491.aspx.

Чаще всего плитка имеет заголовок и подзаголовок, состоящий, возможно, из нескольких строк. Но есть также ряд шаблонов, к которым добавлено изображение.

При выборе шаблона нужно также брать в расчет размер плитки и предусматривать желание пользователя изменить размер шаблона «на лету», используя для этого контекстное меню плитки. Для данного упражнения нужно выбрать следующие шаблоны, соответственно, для больших и малых плиток:

```
Windows.UI.Notifications.TileTemplateType.tileWideText01
Windows.UI.Notifications.TileTemplateType.tileSquareText02
```

Первый шаблон состоит из четырех строк текста разного стиля. Первая строка выводится крупным шрифтом, а для следующих строк используется обычный размер шрифта, они выводятся на разных строках без разрывов текста. Второй шаблон предназначен для малых плиток и состоит из одной строки заголовка, выводимой крупным шрифтом. За строкой заголовка следует подзаголовок, выводимый обычным шрифтом и занимающий максимум три строки.

Чтобы задействовать выбранные шаблоны плиток, добавьте в файл liveTiles.js следующий код:

```
liveTilesManager.enable = function (listOfTasks) {
    // Подготовка шаблона для БОЛЬШОЙ плитки
    var templateLarge =
      Windows.UI.Notifications.TileTemplateType.tileWideText01;
    var xmlLarge =
      Windows.UI.Notifications.TileUpdateManager.getTemplateContent
        (templateLarge);
    var textElementsLarge = xmlLarge.getElementsByTagName("text");

    // Подготовка шаблона для МАЛОЙ плитки
    var templateSmall =
      Windows.UI.Notifications.TileTemplateType.tileSquareText02;
    var xmlSmall =
      Windows.UI.Notifications.TileUpdateManager.getTemplateContent
        (templateSmall);
    var textElementsSmall = xmlSmall.getElementsByTagName("text");

    // Сюда помещается остальной код
}
```

Из предыдущего упражнения понятно, что шаблон плитки состоит из нескольких текстовых элементов. Имеющиеся в коде переменные `textElementsLarge`

и `textElementsSmall` являются массивами, состоящими из XML-узлов, ссылающихся на текстовые элементы в двух XML-шаблонах.

Следующий шаг состоит из заполнения этих текстовых элементов данными, принадлежащими запущенному приложению.

Выбор данных для вывода на плитках

Приложение TodoList полностью основано на списке заданий. У каждого задания есть описание, срок выполнения и приоритет. На живой плитке приложения TodoList, скорее всего, должно выводиться самое последнее задание или следующее задание, требующее завершения. Функция `liveTilesManager.enable` получает список текущих заданий и решает, какую информацию выводить.

В данном упражнении мы выберем первое задание и визуализируем его описание и срок выполнения. Для этого в файл liveTiles.js нужно добавить следующий код, точнее, этот код нужно добавить в конец функции `liveTilesManager.enable`:

```
// Получение информации из приложения
 // для плитки (плиток) с целью ее отображения
 var featuredTask = listOfTasks.getAt(0);

 // Добавление данных к плитке (плиткам)
 textElementsLarge[0].innerText = "TO DO";
 textElementsLarge[1].innerText = featuredTask.description;
 textElementsLarge[2].innerText = "";
 textElementsLarge[3].innerText = "due by</b>: " +
 featuredTask.dueDate.toLocaleDateString();

 textElementsSmall[0].innerText =
   liveTilesManager.getDueDateCompact(featuredTask);
textElementsSmall[1].innerText = featuredTask.description;
```

Также нужно добавить код к функции `liveTilesManager.getDueDateCompact`. Эта функция является вспомогательной, выполняя простое преобразование срока выполнения в формат *мм/дд/гггг*. Ее нужно поместить в конец файла liveTiles.js:

```
liveTilesManager.getDueDateCompact = function (task) {
   var date = task.dueDate;
   var day = date.getDate();
   var month = date.getMonth();
   month++;
   var year = date.getFullYear();
   var x = month + "/" + day + "/" + year;
   return x;
}
```

Объединение шаблонов малых и больших плиток

Несмотря на отсутствие на этот счет строгого требования, в любом приложении Магазина Windows нужно предусмотреть поддержку как малых, так и больших

плиток. До сих пор обе плитки настраивались независимо друг от друга, но этого недостаточно. Windows 8 требует, чтобы большие и малые плитки были объединены в один шаблон. Это можно сделать программно, дополнив функцию `liveTilesManager.enable` следующим кодом:

```
// Объединение шаблонов малых и больших плиток
 var node = xmlLarge.importNode(xmlSmall.getElementsByTagName("binding").
   item(0), true);
xmlLarge.getElementsByTagName("visual").item(0).appendChild(node);
```

В результате выполнения этого кода шаблон малой плитки добавится к шаблону большой плитки. Теперь все готово к созданию из шаблона объекта уведомлений и его регистрации в системе. Для этого нужен такой код:

```
// Создание объекта уведомлений
 var tileNotification =
   new Windows.UI.Notifications.TileNotification(xmlLarge);
 Windows.UI.Notifications
   .TileUpdateManager.createTileUpdaterForApplication().
    update(tileNotification);
```

Итак, все подготовительные операции, касающиеся создания плиток, выполнены. Теперь осталось связать приложение с плитками.

Связывание плиток и приложения

Плитки приложения обновляются, когда выполнение кода приложения доходит до инструкции обновления объекта уведомлений. Частота этих обновлений и выводимый контент зависят от приложения. Что касается приложения TodoList, объект уведомлений создается после запуска и обновляется при каждом редактировании, удалении или создании нового задания. Тем самым гарантируется, что пользователю в качестве напоминания всегда будут предоставляться самые свежие данные, даже когда приложение не выполняется.

Учитывая структуру приложения TodoList, лучше всего функцию `liveTilesManager.enable` вызывать из функции `populateTaskList`, которая, как вы знаете, определена в файле todolist.js. Найдите эту функцию и внесите в нее следующие изменения:

```
TodoList.populateTaskList = function () {
   var promise = new WinJS.Promise(function (complete) {
     var tasks = new Array();
     var localFolder = Windows.Storage.ApplicationData.current.roamingFolder;
     localFolder.getFilesAsync()
       .then(function (files) {
         var io = Windows.Storage.FileIO;
         files.forEach(function (file) {
           io.readTextAsync(file)
             .then(function (json) {
                var task = TodoList.deserializeTask(json);
                tasks.push(task);
                })
```

```
            .then(function () {
             var tasksList = new WinJS.Binding.List(tasks);
             tasksList = tasksList.createSorted(function (first,
                 second) {
                 return first.dueDate > second.dueDate;
            });
            var listview = document.getElementById("task-listview").
                 winControl;
             listview.itemDataSource = tasksList.dataSource;

            // Уведомление
            liveTilesManager.enable(tasksList);
          });
        });
    })
  });
  return promise;
}
```

По сравнению с исходной версией, в функции `populateTaskList` произошли два значимых изменения. Первое заключается в явном вызове функции `liveTilesManager.enable`, включающей живые плитки. А второе связано с объектом обязательства (`promise`), в который заключено все тело функции. Объект `promise` не является строго обязательным, но помогает в случае продолжения разработки. Когда функция `populateTaskList` возвращает объект `promise`, это дает возможность объединить поведение `populateTaskList` с другим поведением посредством методов `then` и `done`.

Большая живая плитка приложения показана на рис. 13.8.

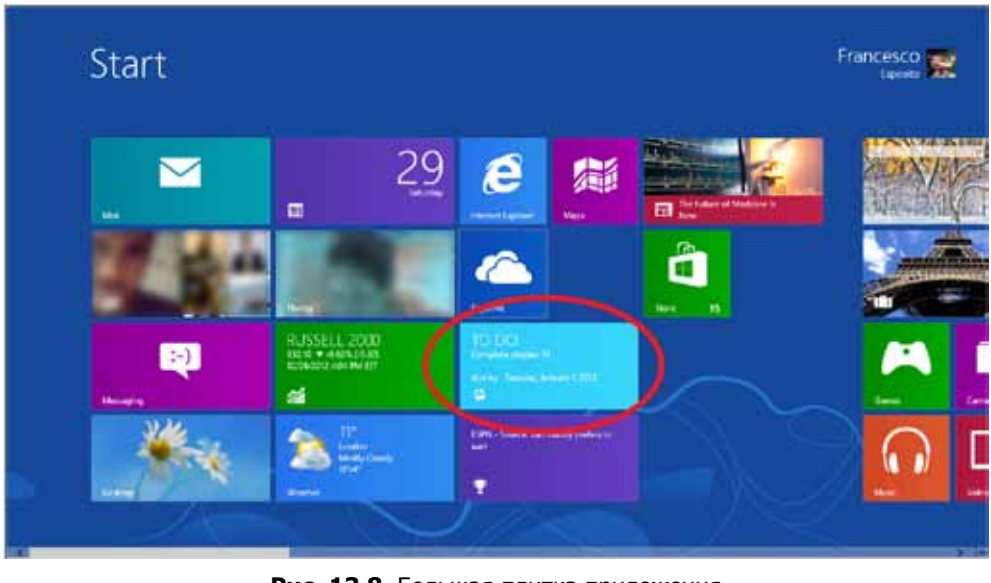

Рис. 13.8. Большая плитка приложения

Малая плитка показана на рис. 13.9. Как видите, теперь в контекстном меню плитки приложения появилась команда для включения и выключения режима оперативных уведомлений.

Рис. 13.9. Малая плитка приложения со своим контекстным меню

НЕТРИВИАЛЬНЫЕ ВОЗМОЖНОСТИ ЖИВЫХ ПЛИТОК

В двух упражнениях, рассмотренных в данной главе, представлены основные функциональные возможности живых плиток, чего вполне достаточно, чтобы разработчик мог приступить к программированию плиток. Но перечень функциональных возможностей плиток этим не исчерпывается. В частности, можно задать срок отключения режима уведомлений, чтобы живые плитки перестали обновляться, как только наступит заданное время. Это особенно полезно, когда приложению нужно выводить данные в форме напоминаний. Кроме того, содержимое плиток можно связать с фоновыми агентами, чтобы оно обновлялось в фоновом режиме, даже когда приложение не запущено. Фоновый агент представляет собой фрагмент кода, у которого нет пользовательского интерфейса, но который периодически запускается под управлением операционной системы. Благодаря фоновому агенту приложение может извлекать данные в асинхронном режиме и обновлять плитки даже без визуализации собственного пользовательского интерфейса.

Выводы

В этой главе мы узнали о живых плитках. Живые плитки — это уникальный инструмент Windows 8, которым приложения могут воспользоваться для вывода информации непосредственно на начальный экран. Плитка по умолчанию назначается любому приложению Магазина Windows во время его установки, а с помощью специального кода плитку можно оживить, заставив получать и визуализировать информацию приложения.

В первом упражнении мы сконфигурировали учебное приложение, снабжающее малую плитку заранее заготовленными данными. Затем мы доработали приложение TodoList из главы 10, включив в него малую и большую живые плитки с целью вывода деталей, касающихся последнего задания. Пользователь может управлять плитками: в любое время он может переключаться между малой и большой плитками и даже полностью отключить режим оперативных уведомлений.

Этой главой заканчивается изучение темы программирования для Windows 8, и теперь все готово для публикации приложения в Магазине Windows.

14 Публикация приложения

Успех — это умение двигаться от неудачи к неудаче, не теряя энтузиазма.

Уинстон Черчилль

Многие годы публикация Windows-приложения ограничивалась созданием программы установки, возможно, с прицелом на специальную платформу. В конечном счете программа установки во многом похожа на хорошо продуманный сценарий, согласно которому файлы копируются в целевую папку, подготавливаются базы данных, конфигурируется система и создается ярлык на рабочем столе. Важно отметить, что программа установки могла распространяться автором без каких-либо посредников. Кроме того, автор сам отвечал за рекламу приложения.

Для мобильных платформ, таких как iOS и Windows Phone, подход к проблеме распространения приложений изменился. Владелец платформы — Apple для iOS и Microsoft для Windows Phone — берет на себя ответственность за распространение и за более высокую степень доступности приложений, создав для этих целей централизованный магазин. Аналогичный подход взят на вооружение в родных для Microsoft Windows 8 приложениях, которые фактически всем известны как приложения Магазина Windows.

В Windows 8 приложения, имеющие родной пользовательский интерфейс, а именно с такими приложениями мы и работали в книге, могут распространяться только через Магазин. Главной причиной такого выбора способа распространения является гарантия хорошего качества и хорошей работы с устройствами, а также отсутствие ошибок и уязвимостей в системе безопасности. Компания Microsoft в процессе

сертификации убеждается в приемлемости приложения для Магазина, но, как вы уже, наверное, догадались, сертификация связана с определенными затратами. Затраты составляют 30 % от стоимости платных приложений. Что касается бесплатных приложений, существует ограничение на количество приложений, выкладываемое в Магазин, и при превышении порога выплачивается твердая ставка.

В данной книге мы научились создавать приложения Магазина Windows с помощью нескольких новых доступных прикладных программных интерфейсов. Теперь самое время завершить цикл обучения и научиться публиковать разработанные приложения в Магазине.

Получение учетной записи разработчика

Итак, пользователи Windows 8 могут устанавливать новые приложения только через Магазин Windows. Но кто может это делать? Любой разработчик приложений для Windows 8 должен получить от Microsoft учетную запись разработчика, которая даст право публиковать приложения. Не имея действующей учетной записи разработчика, ваше замечательное приложение никогда никуда не попадет, исключая компьютер, на котором оно разрабатывалось.

ПРИМЕЧАНИЕ

В сфере установки приложений Windows 8 предлагает два подхода. Один из них представлен классическим пользовательским интерфейсом Windows, позволяющим установить любое приложения либо самостоятельно, либо посредством предоставляемой программы установки, как и во всех предыдущих версиях Windows. В основе другого подхода лежит современный пользовательский интерфейс Windows 8. Любое приложение подобного рода приобретается и устанавливается только через Магазин Windows.

Регистрация в качестве разработчика бесплатных приложений

Для регистрации в качестве разработчика приложений для Windows 8 нужно в Microsoft Visual Studio открыть меню Store (Магазин), выбрать пункт Open Developer Account (Открыть учетную запись разработчика) и следовать инструкциям, появляющимся на экране (рис. 14.1).

Щелчок на ссылке Get A Developer License For Windows Store Apps (Получить лицензию разработчика приложений для Магазина Windows) приведет вас в Центр разработки Windows 8, где для регистрации в качестве разработчика нужно щелкнуть на кнопке, как показано на рис. 14.2.

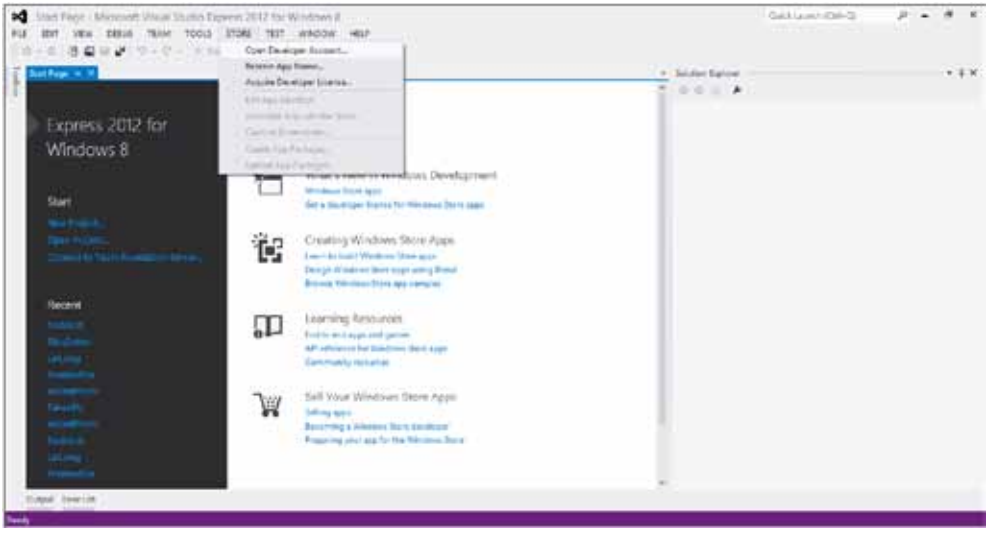

Рис. 14.1. Создание учетной записи разработчика через меню Store

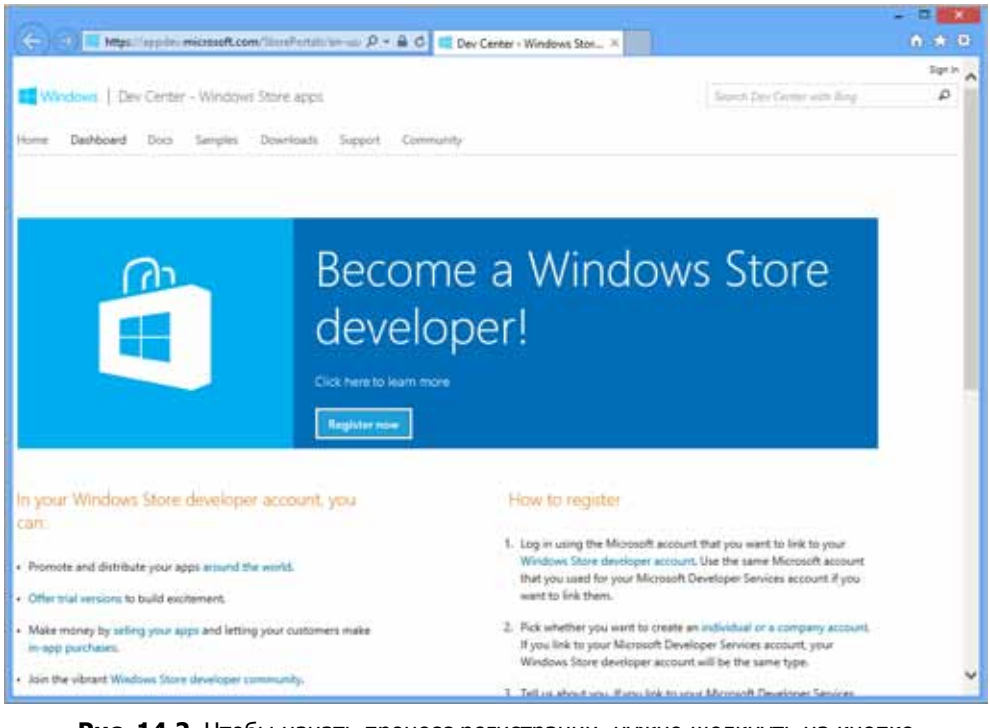

Рис. 14.2. Чтобы начать процесс регистрации, нужно щелкнуть на кнопке

Если у вас уже есть учетная запись разработчика для Windows Phone, ее можно связать с Windows 8 без новой регистрации. Однако чтобы иметь отдельные учетные записи для Windows Phone и Windows 8, требуется предоставить разные идентификаторы. Разумеется, если вы новый разработчик приложений для Windows 8, нужно просто ввести идентификатор и продолжить регистрацию.

Информация об учетной записи

При создании новой учетной записи разработчика нужно знать, какой тип учетной записи вы хотите создать: личный (переключатель Individual) или корпоративный (переключатель Company), как показано па рис. 14.3.

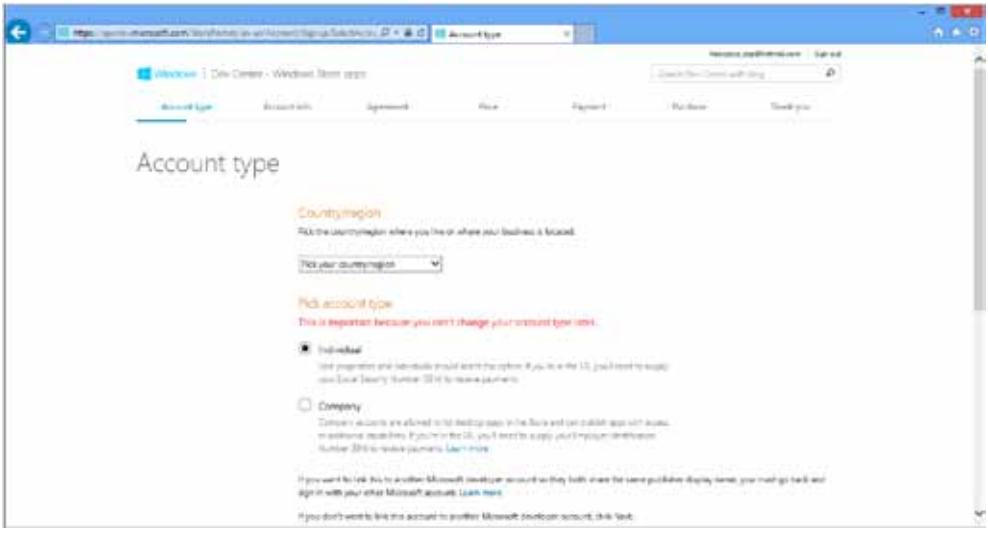

Рис. 14.3. Какая учетная запись вам нужна, личная или корпоративная?

Учтите, что переход с одного типа учетной записи на другой невозможен. Для создания корпоративной учетной записи нужно предоставить компании Microsoft множество дополнительной документации о компании и ее руководителях. В то же время корпоративная учетная запись позволяет иметь несколько администраторов, а также создавать и публиковать приложения для использования внутри компании. Но если вы только практикуетесь в создании приложений для Windows 8 и хотите выложить одно или два разработанных лично вами приложения, то вполне подойдет личная учетная запись.

Продолжите регистрацию и введите свою персональную информацию, включая полное имя и адрес. Затем выберите свое отображаемое имя. Это имя играет важную роль, поскольку именно его Windows 8 будет выводить на экран в качестве имени

автора приложения. Отображаемое имя любого издателя приложений должно быть уникальным. После ввода этого имени сайт Windows 8 Dev Center проведет в реальном времени проверку на доступность или на занятость выбранного вами имени (рис. 14.4).

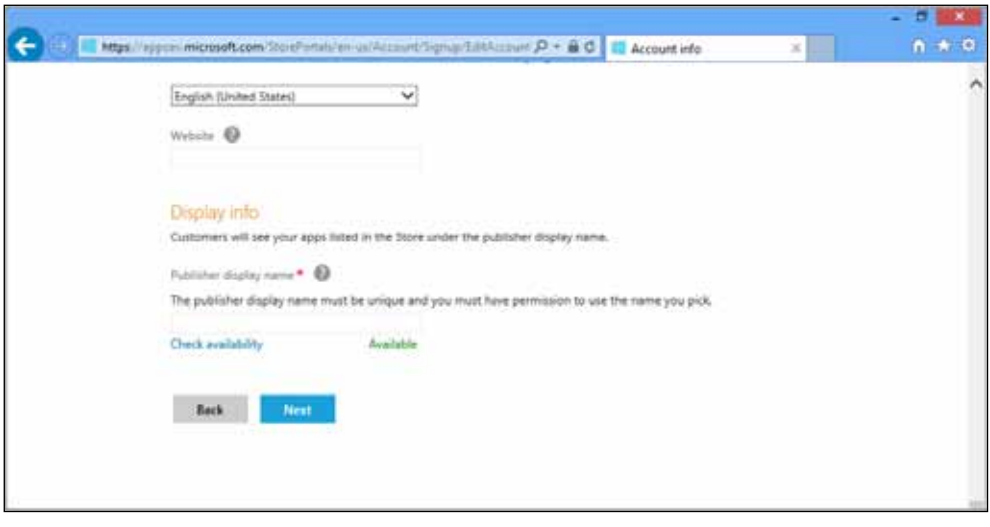

Рис. 14.4. Имя издателя должно быть уникальным

Подробности оплаты

Для создания личной или корпоративной учетной записи вы должны стать зарегистрированным разработчиком приложений для Windows 8. К сожалению, регистрация не бесплатна. Чтобы стать разработчиком приложений для Windows Phone или для iOS, нужно платить, и, чтобы быть разработчиком приложений для Windows 8, нужно также вносить ежегодную плату. Ко времени написания данной книги регистрационная плата для индивидуальных разработчиков составляла 49 долларов, для компаний — 99 долларов (или эквивалентная сумма в других валютах).

После выбора отображаемого имени на экране появится контракт между вами и Магазином Windows. Также появится итоговая страница, на которой изложено все, что вы получите за свои деньги (рис. 14.5). Если вы решите продолжить регистрацию, появится запрос на ввод информации о вашей кредитной карте для внесения платежа. При наличии кода продвижения его также нужно будет ввести.

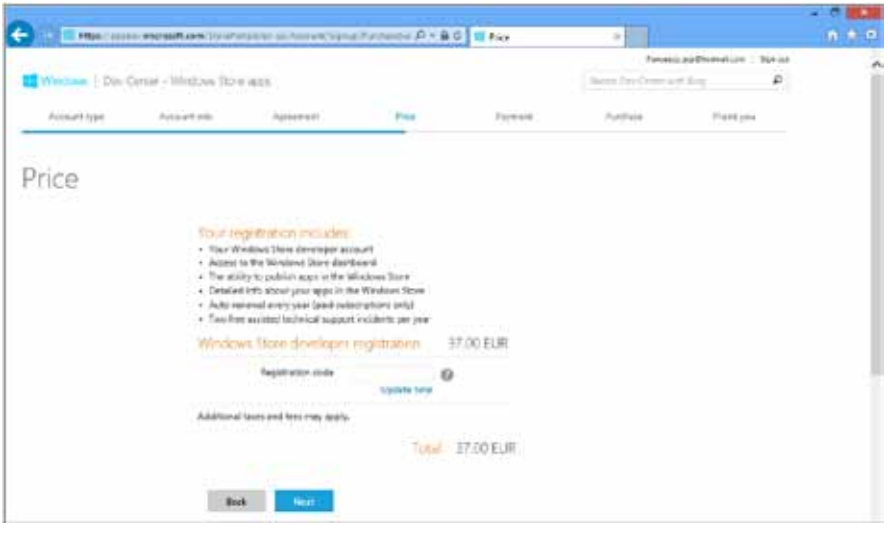

Рис. 14.5. Экран оплаты

Регистрация в качестве разработчика платных приложений

На рис. 14.6 показан конечный экран, появляющийся после успешного завершения регистрации. Теперь все готово для публикации бесплатных приложений. Если же вы собираетесь публиковать платные приложения, нужно пройти еще несколько дополнительных шагов.

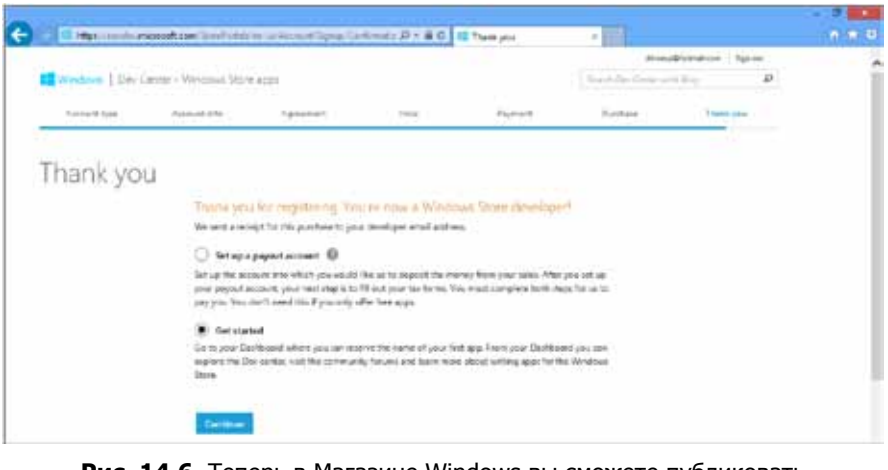

Рис. 14.6. Теперь в Магазине Windows вы сможете публиковать бесплатные приложения

С точки зрения компании Microsoft, основная разница между бесплатными и платными пользовательскими приложениями состоит в том, что в последнем случае компании придется вам заплатить, когда кто-то купит ваши приложения. Следовательно, перед тем как опубликовать ваше приложение в Магазине, компании Microsoft нужно выяснить, как затем она будет с вами расплачиваться за продажи.

Вам нужно предоставить два отдельных блока информации: банковские реквизиты для перевода денег и налоговую информацию для отчета перед налоговыми органами. Банковская информация может меняться в зависимости от страны, в которой вы живете, но чаще всего в ней требуется только международный банковский код и буквенно-цифровая строка (IBAN-строка), идентифицирующая ваш лицевой счет.

Если нужно решить налоговые вопросы, придется заполнить электронную форму, имеющую разное содержание для резидентов и нерезидентов США. Детали прохождения этого этапа могут изменяться от страны к стране, но в большинстве случаев посылать по обычной почте какие-нибудь бумажные документы вам не придется. Ввод налоговых и банковских данных — обычно весьма нудная процедура, но в данном случае она не займет слишком много времени, и будем надеяться, выполнить ее придется только один раз. Дополнительные сведения можно найти в инструкции, пройдя по ссылке http://bit.ly/PrOrbW.

Этапы публикации приложения

Все приложения Магазина Windows попадают в один и тот же каталог, поэтому у каждого приложения должно быть уникальное имя.

Выбор имени для приложения

В силу упомянутых ранее обстоятельств выбор имени может стать довольно важным шагом. Если вы придумали какое-то конкретное имя, то его нужно заранее зарезервировать. Если имя, которое вам понравилось, уже занято, ничего не остается, как выбрать другое имя!

Резервирование имени приложения

Имя для будущего приложения можно зарезервировать в любое время, выбрав в Visual Studio команду Reserve App Name (Зарезервировать имя приложения) в меню Store (Магазин), как показано на рис. 14.7.

Чтобы зарезервировать имя приложения, его нужно ввести. Если Магазин Windows не обнаружит конфликтов, будет получен положительный ответ (рис. 14.8).

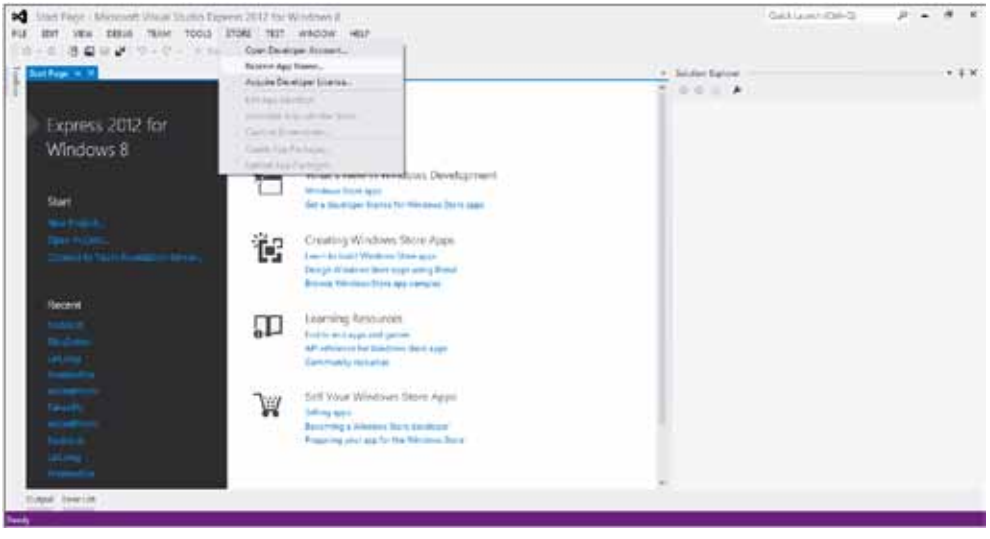

Рис. 14.7. Начало процедуры резервирования имени приложения

Рис. 14.8. Имя T-List зарезервировано

Учтите, что, когда имя зарезервировано, пользоваться им сможете только вы сами. Но резервирование имени действует только один год; если вы не опубликуете приложение под этим именем в течение года, пока действует резервирование, Магазин Windows отменит резервирование и именем смогу воспользоваться другие.

Локализация имени приложения

Иногда имя, данное приложению, является «нейтральным» по отношению к разговорным языкам. В таких случаях вам не придется менять имя независимо от

количества языков, для которых нужно локализовать приложение. Но это довольно редкая ситуация. Чаще приходится переводить или просто менять имя приложения, адаптируя его к языку целевой аудитории. Рекомендуется использовать разные имена для каждого из поддерживаемых языков, но это не обязательно. Если вы планируете локализацию имен, то продолжение показано на рис. 14.9.

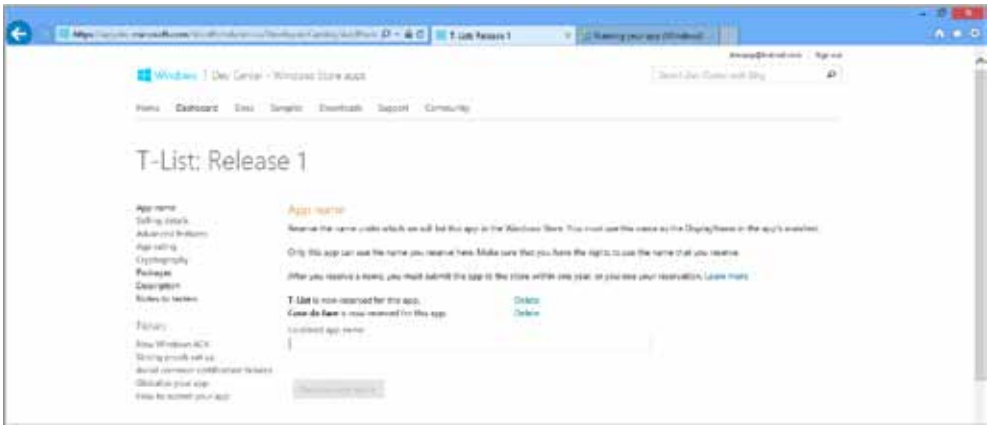

Рис. 14.9. Выбор локализованных имен

После резервирования имени можно приступать к созданию пакета приложения.

ПРИМЕЧАНИЕ

Как уже упоминалось, если имя действительно имеет для приложения важное значение, то лучше сначала зарезервировать имя, а уже потом приступать к разработке приложения.

Создание пакета приложения

Этот шаг состоит в создания единого контейнера со всеми файлами вашего приложения. Перед тем как приступить к созданию пакета, нужно убедиться в функциональной завершенности приложения и в отсутствии ошибок и уязвимостей.

Компилирование в режиме выпуска с использованием логотипа logo.png

В завершенном виде приложение Магазина Windows не требует дальнейших изменений кода и включает в себя определенные файлы для различных логотипов и начальной заставки. Кроме того, может также потребоваться наличие нескольких

копий экрана, способствующих рекламе приложения в Магазине Windows. Файлу логотипа нужно уделить особое внимание, поскольку он будет значком вашего приложения в Магазине Windows.

Когда все готово, сначала в режиме выпуска нужно скомпилировать код приложения. Когда приложение компилируется, обычно делается выбор между двумя режимами: режимом отладки (Debug), который выбирается по умолчанию, и режимом выпуска (Release). Оба режима приводят к созданию полноценного исполняемого файла, но в режиме отладки в исполняемый файл внедряется дополнительная информация, используемая в целях отладки. После того как приложение очищено от ошибок и приобрело функциональную завершенность, надобность в отладочной информации отпадает. Компиляция в режиме выпуска позволяет получить более компактный (и даже чуть более быстрый) исполняемый файл, в котором нет лишних отладочных символов.

Подготовка пакета приложения

Получив исполняемый файл, скомпилированный в режиме выпуска, можно продолжить создание пакета приложения. Пакет создается выбором команды Create App Packages (Создать пакет приложения) в меню Store (Магазин), как показано на рис. 14.10.

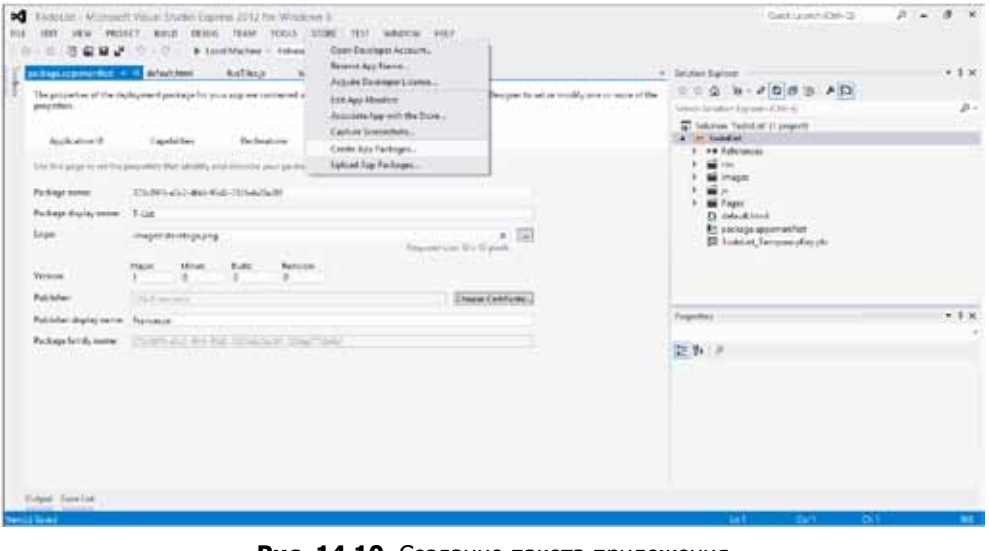

Рис. 14.10. Создание пакета приложения

Для создания пакета нужно выполнить ряд шагов, в ходе этой процедуры Visual Studio выведет на экран несколько форм, которые вам потребуется заполнить. Первая форма показана на рис. 14.11. Эта форма требует ввода имени публикуемого приложения. В форме перечисляются все зарезервированные имена, если таковые у вас имеются. Можно выбрать одно из этих имен или запустить Мастер для резервирования на данном этапе нового имени.

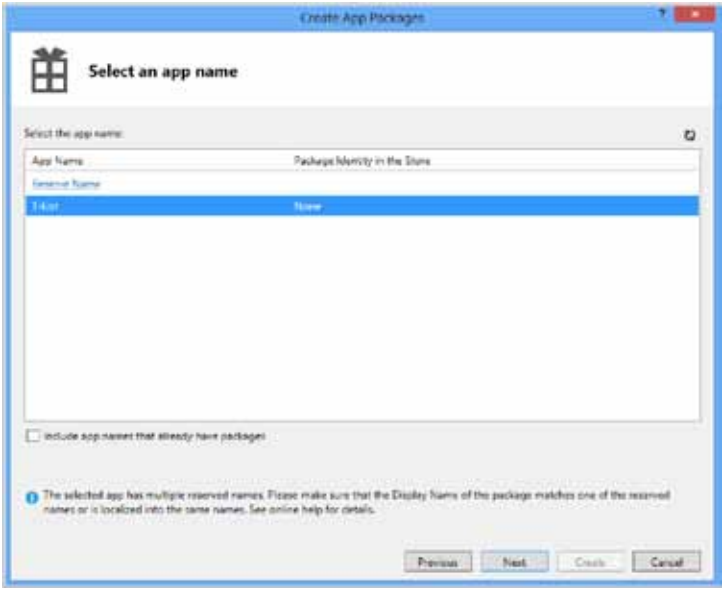

Рис. 14.11. Задание имени приложения

На следующем шаге выбирается целевая папка на локальном диске, где будет сохранен пакет. Нужно обязательно запомнить эту папку, поскольку она понадобится в самом конце процедуры создания пакета, чтобы выложить пакет в Магазин Windows. Также нужно выбрать как минимум одну целевую платформу в форме, показанной на рис. 14.12.

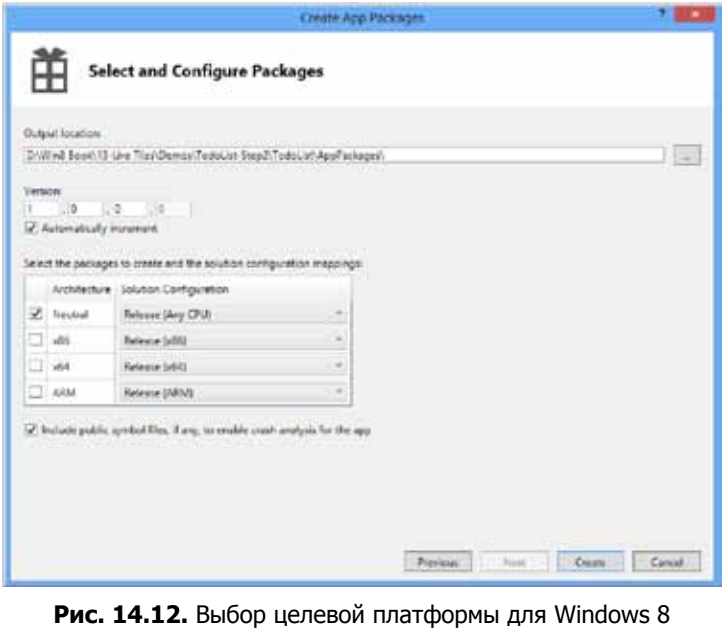

Рис. 14.12. Выбор целевой платформы для Windows 8

Целевая платформа определяет аппаратную архитектуру машины, на которой может запускаться ваше приложение. Чтобы гарантировать для вашего приложения как можно более широкую аудиторию, обычно выбирают все платформы.

ВНИМАНИЕ

Смысл здесь в том, что операционная система Windows 8 запускается на множестве различных аппаратных конфигураций, чаще всего на 32-разрядных машинах (x86), 64-разрядных машинах (x64) и таких устройствах, как планшетный компьютер с архитектурой ARM. Чтобы стало понятнее, если не установить флажок ARM, то ваше приложение Магазина Windows не сможет работать на планшетных устройствах с Windows 8.

Когда в форме, показанной на рис. 14.12, производится щелчок на кнопке Create (Создать), Visual Studio начинает процедуру создания полноценного пакета для приложения Магазина Windows. В ходе этого процесса Visual Studio запускает приложение и выполняет в отношении него ряд тестов. Если проверка всего проходит успешно, появляется диалоговое окно, показанное на рис. 14.13.

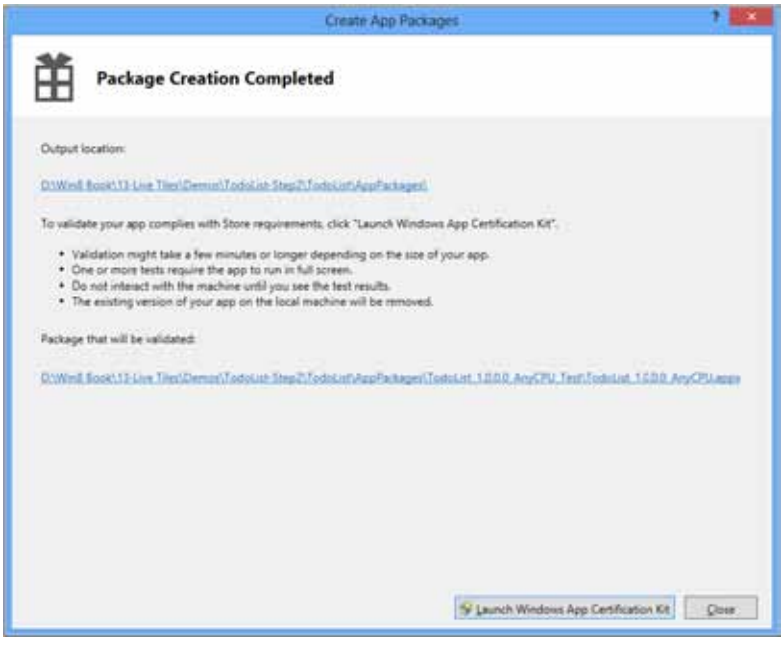

Рис. 14.13. Пакет приложения успешно создан

Наличие готового пакета еще не означает, что все закончено. Приложение пока не готово к выкладыванию в Магазин Windows. Создан только лишь пакет в формате, который может быть принят Магазином Windows. Однако нет никакой гарантии, что ваше приложение в этом виде будет принято и опубликовано в магазине. Перед тем как появиться в Магазине Windows, приложение должно быть сертифицировано.

Любое приложение, отправленное в Магазин Windows, требует для публикации прохождения ряда дополнительных тестов. Если приложение независимо от причины не пройдет любой из этих тестов, Магазин Windows забракует приложение, предоставив документ об отклонениях в поведении приложения и ряд рекомендаций по их устранению.

Комплект сертификации Windows-приложений

Чтобы исключить отбраковку приложения или, точнее, снизить вероятность его отбраковки, набор проверочных тестов, предлагаемых компанией Microsoft, можно запустить на локальной машине еще до отправки приложения в Магазин. Это делается с помощью комплекта сертификации Windows-приложений (Windows App Certification Kit). Как показано на рис. 14.13, можно щелкнуть на кнопке Launch Windows App Certification Kit в нижней части экрана и локально провести те же самые тесты, на которых будет испытываться ваше приложение после его отправки. Двухэтапную процедуру иллюстрирует рис. 14.14.

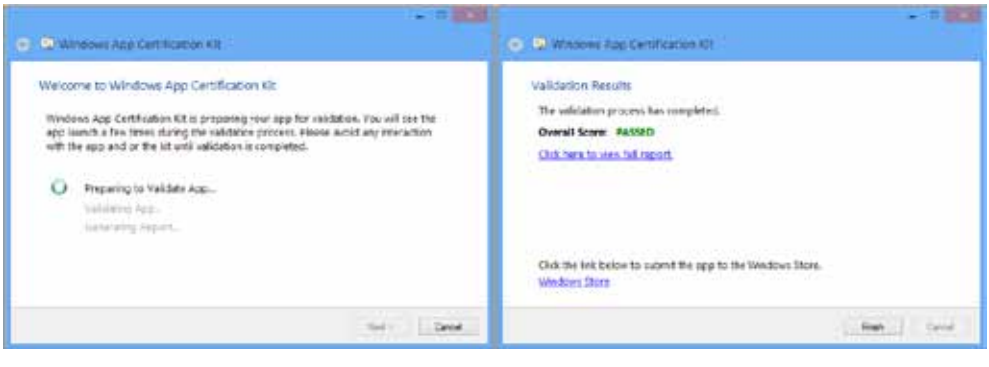

Рис. 14.14. Выполнение тестов с помощью комплекта сертификации Windows-приложений

Чтобы узнать подробности тестирования, можно посмотреть отчет, щелкнув на ссылке в последнем диалоговом окне. Примерный отчет для прошедшего тесты приложения показан на рис. 14.15.

Прохождение тестов с помощью комплекта сертификации Windows-приложений не дает гарантии, что ваше приложение будет отвечать всем критериям Магазина Windows и будет принято. Но шансы на его публикацию несомненно возрастут.

ПРИМЕЧАНИЕ

Перед отправкой приложения лучше всегда проверять его с помощью комплекта сертификации Windows-приложений. После отправки может пройти целая неделя, прежде чем вы получите ответ от компании Microsoft. При отрицательном ответе, даже если предположить, что вы тут же решите проблему и снова отправите приложение в Магазин, потребуется минимум еще одна неделя, пока ваше приложение будет опубликовано.

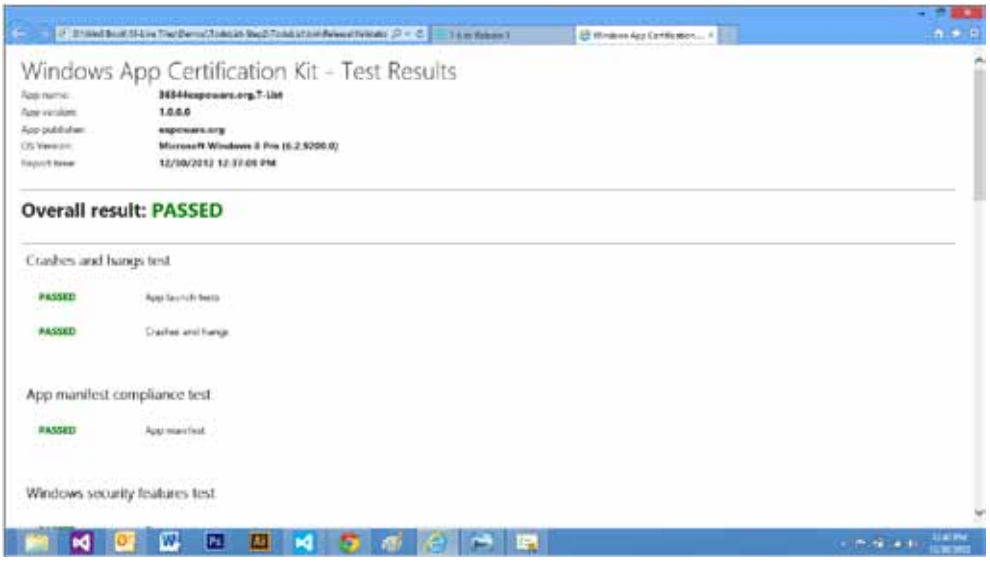

Рис. 14.15. Пример отчета при успешном прохождении приложением стандартных тестов

Выгрузка приложения

Завершающим шагом в публикации приложения является отправка пакета компании Microsoft. Это делается выбором в Visual Studio команды Upload App Packages (Выгрузить пакет приложения) в меню Store (Магазин). Когда начнется передача пакета, появится страница, похожая на ту, что показана на рис. 14.16.

Как только пакет попадет на серверы Магазина Windows, нужно будет дать соответствующее описание вашего приложения, чтобы подготовить его для финального тестирования. На данном этапе нужно ввести описание, предназначенное для пользователей, и какие-нибудь ключевые слова, призванные помочь им найти ваше приложение в магазине (рис. 14.17). Здесь же можно добавить копии экрана, помогающие продать ваше приложение потенциальным покупателям.

Перед публикацией приложение пройдет еще несколько тестов. Некоторые из них проводятся в автоматическом режиме (например, тесты сертификации Windows-приложений), другие выполняет сотрудник компании. Чтобы этот сотрудник быстрее понял, как работает ваше приложение, можно ввести для него пояснительную информацию. Как показано на рис. 14.18, добавление пояснений для тестеров является завершающим шагом перед фактической отправкой приложения на сертификацию.

После отправки приложения на сертификацию пакет приложения попадает в очередь, ожидая прохождения автоматизированных тестов. Эти тесты призваны проверить приложение на безопасность и на соответствие техническим требованиям.

Они аналогичны сертификационным тестам, которые вы должны были провести на своей локальной машине, а поскольку они проводятся в автоматическом режиме, то уже через несколько часов вы будете знать, прошло ли их ваше приложение или было забраковано. В случае неблагоприятного развития событий вы тут же получите уведомление.

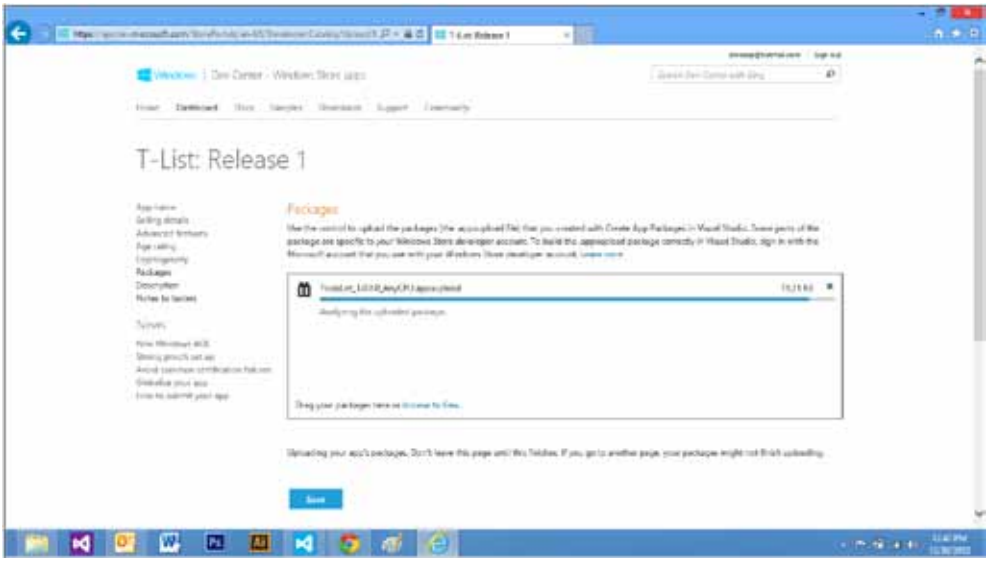

Рис. 14.16. Выгрузка пакета T-List в Магазин Windows

Рис. 14.17. Ввод описания и ключевых слов для поиска вашего приложения

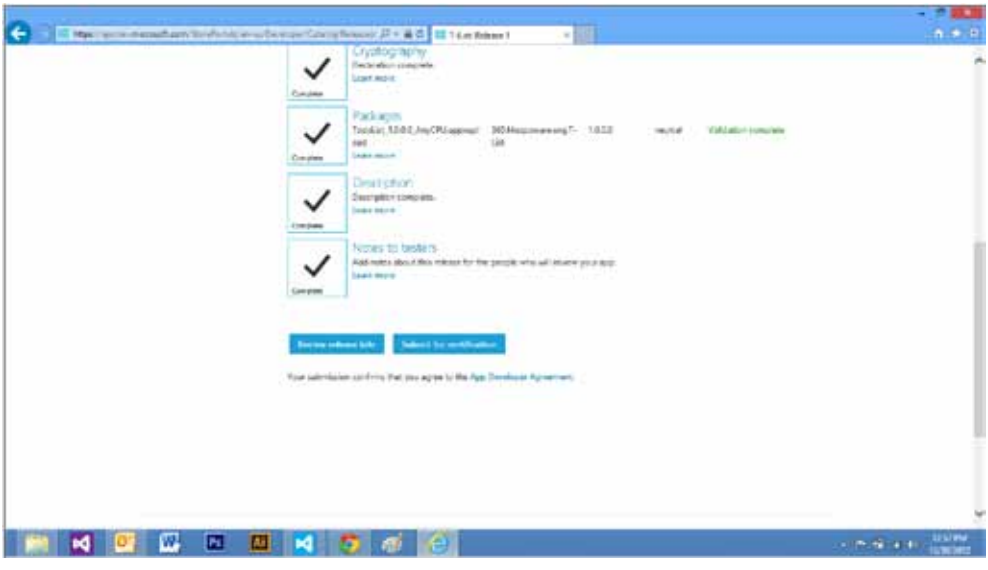

Рис. 14.18. Все сделано, и приложение готово к сертификации

Наиболее деликатной частью сертификации является проверка информационного наполнения. Эта задача решается специалистом, что может потребовать до семи дней. После прохождения этого теста приложение переходит в режим выпуска. При вводе описания приложения вы также могли принять решение о процедуре выпуска. Выпуск приложения может быть осуществлен после прохождения всех тестов или в ближайшую удобную для вас дату. После того как приложение принято и вы готовы к его публикации, пакет отправляется на процедуру подписания, где двоичный код подписывается ключом, предотвращающим вмешательство, а затем ставится в выходную очередь, чтобы в конце концов появиться на серверах Магазина Windows.

Приложения только для корпоративных пользователей

Только что рассмотренный сценарий выполняется чаще всего. Но есть и другой сценарий, о котором стоит упомянуть. Что, если вы не собираетесь публиковать приложение в Магазине, а хотите открыть к нему доступ только для хорошо известного покупателя или покупателей?

Публикация приложения в Магазине Windows, будь она бесплатной или платной, дает любому владельцу машины, оснащенной Windows 8, возможность загрузить и установить ваше приложение. Это, конечно, желательно в большинстве, но не во всех случаях.

Поэтому Windows 8 предлагает другой способ распространения приложений, известный как публикация только для корпоративных пользователей (Enterprise

Sideloading). Не все машины с Windows 8 обеспечивают поддержку таких приложений, поэтому, становясь на этот путь, вы ограничиваете свою целевую аудиторию только владельцами корпоративной версии (Enterprise edition) Windows 8. Большинство базовых версий Windows 8 не могут использовать приложения с ограниченной публикацией, а поддерживают только приложения, загружаемые и устанавливаемые из Магазина. Тем не менее даже обычные машины с Windows 8 смогут использовать приложения с ограниченной публикацией после приобретения специальных ключей, предназначенных для применения только на одной машине.

Кроме того, для работы приложений с ограниченной публикацией целевые машины должны быть частью домена, на котором включена политика Allow all trusted apps to install (Разрешить установку всех доверенных приложений).

Выводы

Есть всего три варианта написания приложений, выполняемых на машине под управлением Windows 8. Первый вариант заключается в написании приложений рабочего стола Windows, которые также выполняются на более ранних версиях Windows. Но если выбрать этот вариант, то наша книга вам не нужна. Фактически, такие приложения пишутся в среде .NET Framework с использованием в качестве основного языка программирования C# или Visual Basic.

Второй вариант, который послужил темой данной книги, касается написания приложений Магазина Windows с современным пользовательским интерфейсом. Но эти приложения обычно предназначены для распространения через Магазин Windows безо всяких ограничений. Это означает, что после того, как приложение становится доступным в Магазине Windows, любой пользователь Windows 8 может его получить и установить.

И наконец, третий вариант заключается в создании приложений, публикуемых только для корпоративных пользователей. У этих приложений современный пользовательский интерфейс Windows 8, который рассматривался в данной книге, но их можно устанавливать только на машинах, настроенных для выполнения доверенных приложений. Важно отметить, что приложение, публикуемое только для корпоративных пользователей, ничем не отличаются от приложения Магазина Windows, но его целевая среда должна быть сконфигурирована так, чтобы позволить установку приложений из мест, отличных от Магазина Windows, например через электронную почту, с CD- или DVD-диска.

В этой книге мы рассмотрели все вопросы разработки приложений для Windows 8 средствами HTML5 и JavaScript, начиная от создания простейшего приложения «Hello Windows 8» и заканчивая публикацией бесплатных и платных приложений. Если вы добрались до этого места и выполнили все упражнения, значит, вы полностью готовы к публикации совершенно выдающихся приложений для Windows 8. Удачи!