БИБЛИОТЕКА ПРОГРАММИСТА

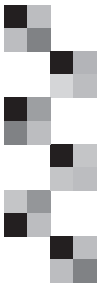
Дж. Ликнесс

Приложения для Windows 8 на C# и XAML

- WinRT — новая среда выполнения Windows
- Приложения для Windows 8
- MVVM и тестирование
- Пакетирование и развертывание

Addison-Wesley

ПИТЕР



Building Windows 8 Apps with C# and XAML

Jeremy Likness

Addison-Wesley

Upper Saddle River, NJ • Boston • Indianapolis • San Francisco
New York • Toronto • Montreal • London • Munich • Paris • Madrid
Capetown • Sydney • Tokyo • Singapore • Mexico City

БИБЛИОТЕКА ПРОГРАММИСТА

Дж. Ликнесс

Приложения для Windows 8 на C# и XAML

Москва · Санкт-Петербург · Нижний Новгород · Воронеж
Ростов-на-Дону · Екатеринбург · Самара · Новосибирск
Киев · Харьков · Минск
2013

ББК 32.973.2-018.2
УДК 004.451
Л56

Ликнесс Дж.
Л56 Приложения для Windows 8 на C# и XAML. — СПб.: Питер, 2013. — 368 с.: ил. — (Серия «Библиотека программиста»).

ISBN 978-5-496-00349-0

Это первое практическое руководство по созданию приложений для Windows 8 охватывает весь жизненный цикл: от разработки шаблона проекта до публикации в Windows Store. Автор книги Джереми Ликнесс, ведущий специалист компании Microsoft, поможет использовать ваши навыки разработчика в работе с новыми инструментами Visual Studio 2012 для создания полезных и инновационных приложений. В книге рассмотрены и бизнес-приложения, и пользовательские приложения. При помощи тщательно отработанных загружаемых примеров кода и демонстрационных проектов автор показывает, как максимально использовать новые функции платформы, включая встраивание в социальные сети, поиск, расширения, контракты и мозаичное размещение.

ББК 32.973.2-018.2
УДК 004.451

ISBN 978-0321822161 англ.
ISBN 978-5-496-00349-0

Краткое содержание

Содержание

Моей маме. Твои поддержка и ободрение всегда были для меня благословением. Мне грустно от того, что я не могу рассказать тебе о том, что книга написана.

Предисловие

Жизнь разработчика программного обеспечения нелегка. Примерно каждые десять лет он должен забыть все, что знает, и приняться за учебу. Времена меняются, а технологии меняются еще быстрее. Десять лет назад программистам пришлось переучиваться, чтобы перейти с Win32 на .NET и C#. Сегодня настало время новой платформы. Она называется Windows 8. Вместе с ней пришли глубокие изменения и в принципы создания программ для Windows, и в то, как они исполняются системой.

Windows 8 не похожа на те операционные системы семейства Windows, которые мир видел раньше. Новая модель программирования для Windows, прежде всего, ориентирована на простоту и безопасность работы, на эффективность расходования устройствами энергии батарей. Современные приложения для Windows работают в полноэкранном режиме и исполняются по одному, поэтому им, по большей части, не приходится делить системные ресурсы с другими программами. Их пользовательский интерфейс может быть построен на основе технологии XAML, HTML или DirectX. Эти программы работают в изолированной от остальной системы среде, в так называемой «песочнице», которая предотвращает выполнение вредоносного кода. Кроме того, их проверяют перед публикацией в Магазине Windows для того, чтобы убедиться, что они не нарушают правил «песочницы». Приложения для Windows 8 ориентированы на сенсорные экраны, но с ними, с тем же успехом, можно взаимодействовать с помощью мыши и других устройств ввода. Более того, они устанавливаются одним щелчком, а после их удаления не остается никаких следов.

За новым пользовательским интерфейсом стоит новый прикладной программный интерфейс (API), называемый Windows Runtime API (API среды выполнения Windows), но более известный как WinRT. WinRT — это новый взгляд на API для Windows. Предыдущий прикладной программный интерфейс для Windows устарел, к тому же он был чрезмерно сложен

и привязан к определенному языку. API WinRT, напротив, является совершенно современным и обращаться к нему можно посредством множества языков. Одна из замечательных особенностей Windows 8 заключается в том, что разработчики, которые используют HTML и JavaScript, имеют те же возможности, что и программисты, пишущие на XAML и C#.

Для создателей программ, как вы можете догадаться, это означает, что пришло время все начать сначала. Теперь WinRT — *это* API для Windows. Кроме того, это новый инструмент создания пользовательского интерфейса, ранее известного как интерфейс «Метро». Подобный интерфейс — лицо приложений для Windows. Наберитесь смелости, чтобы осваивать новые технологии, или останетесь на обочине.

Для того чтобы стать разработчиком программ для Windows 8, нужно изучить WinRT. То есть приобрести прочные навыки асинхронного программирования, понимать особенности жизненного цикла приложения, знать, что приложение для Windows, которое не видно пользователю, приостанавливается, а приостановленное приложение может быть в любое время без предупреждения полностью остановлено системой. Разработчик для Windows 8 умеет работать с контрактами, позволяющими приложениям взаимодействовать с чудо-кнопками, панель которых выдвигается с правой стороны экрана. Он понимает и использует возможности активных плиток, push-уведомлений и других механизмов, которые делают приложения полноправными жителями мира Windows 8. Кроме того, разработка качественных программ невозможна без понимания философии дизайна для Windows 8 и того, как с помощью XAML создавать привлекательный, подвижный и чувствительный пользовательский интерфейс.

Если вы оказались в незнакомом месте, хорошо бы для начала найти проводника. Невозможно представить себе лучшего гида по Windows 8, чем Джереми Ликнесс. Из тех, кого я знаю, Джереми — единственный, кто способен работать 32 часа в сутки (еще четыре часа он спит). Да, я встречался с людьми, которые были гораздо умнее меня, но никто из них не был работоспособнее меня. Джереми полностью изменил мое видение данного вопроса. Отправьте ему электронное письмо в 3:00 утра и в 3:02 вы получите ответ. Именно поэтому он работает главным консультантом в Wintellect, и именно поэтому мы предложили ему заняться проектированием и разработкой ориентированных на Windows 8 решений для наших клиентов. Преподаватель не может как следует учить других, если он не имеет практического опыта. Джереми создавал реальные приложения для реальных клиентов. Именно поэтому я от всей души рекомендую эту книгу и с замиранием сердца жду реакции на нее сообщества разработчиков.

Windows 8 — весьма смелый шаг со стороны Microsoft. Возможно, самый смелый со времен появления первой версии этой операционной системы.

Но это — правильный шаг, и сделан он в нужное время. Разработка программного обеспечения в ближайшие десять лет не ограничится традиционными ПК. Она будет направлена на создание приложений для планшетных компьютеров, мобильных телефонов и других подобных устройств. Появятся новые компании, новые миллионеры, и все это случится благодаря приложениям для портативных устройств, среди которых и планшетный компьютер Microsoft Surface. Игнорировать WinRT — значит, игнорировать те инструменты Microsoft, которые позволяют разрабатывать мобильные приложения.

Изучайте WinRT. Двигайтесь вперед, пишите отличные программы. Внесите свой вклад в успех этой платформы. И держите под рукой эту книгу. Когда вы столкнетесь с проблемами, она способна оказать вам помощь, сравнимую с немедленным ответом на электронное письмо в три часа утра. А с точки зрения Джереми, подобная перспектива весьма привлекательна, во всяком случае, она бы избавила его от необходимости вставать ни свет ни заря, чтобы отвечать на такие письма.

Джефф Просис (Jeff Prosise),
один из основателей компании Wintellect

Введение

Первые слухи о Windows 8 появились в начале 2011 года. Множество домыслов гуляло по Интернету, когда разработчики начали интересоваться тем, какой будет новая платформа. Ходили даже слухи, что новая платформа не будет поддерживать .NET Framework, что она будет полностью основана на C++ или на HTML 5 и JavaScript, что на ней не будет работать существующее программное обеспечение. Ранние релизы и экранные снимки появлялись в Твиттере, но это лишь усиливало всеобщие сомнения. Наконец, Стивен Синофски (Steven Sinofsky), президент подразделения Windows в Microsoft, 13 сентября 2011 года представил миру Windows 8.

Я был одним из первых нетерпеливых программистов, которые загрузили дистрибутив системы. Установив Windows 8 на виртуальную машину, я быстро понял, что с поддержкой .NET Framework все в полном порядке. На новой платформе мои Silverlight-приложения работали. Языки C# и XAML присутствовали среди инструментов разработки новых приложений в «Метро-стиле» (название «Метро» в RTM-версии Windows было изменено на «Магазин Windows»). Я не смог присутствовать на конференции в Калифорнии, посвященной выпуску Windows 8, но доклады очень быстро появились в Интернете, и я смотрел их каждый вечер, каждое утро, всегда, когда у меня выдавалась свободная минута.

Windows 8 предоставляет среду выполнения Windows (Windows Runtime) — новую платформу создания приложений, обладающих возможностями, доступными ранее на компьютерах под управлением Windows. Я создавал приложения целыми днями и восхищался тем, что мои знания в области C# и XAML, в использовании Silverlignt и Windows Presentation Foundation (WPF), вполне применимы к новой среде выполнения. Новый набор компонентов этой платформы делает простой как никогда разработку функциональных приложений, ориентированных на сенсорное взаимо-

действие с пользователем. В конце концов, очень скоро я связался с издателем моей книги «Designing Silverlight Business Applications» и сказал, что хочу написать книгу о Windows 8.

К счастью, я участвовал в программе раннего обучения Microsoft. Консалтинговой фирме Wintellect, в которой я работаю, предложили провести практические занятия и семинары, предназначенные для новых разработчиков, желающих научиться создавать приложения для Windows 8. Это позволило мне знакомиться с ранними выпусками продукта и писать о различных функциональных возможностях, которые, в итоге, стали частью финального релиза. Пока я создавал примеры, посвященные сенсорному управлению объектами на экране, передаче форматированного контента между приложениями и использованию на начальном экране «живых» интерактивных плиток с возможностью быстрого просмотра, новая система нравилась мне все больше и больше.

В рамках работы над этой книгой я написал статью о десяти главных причинах, по которым разработчикам понравится создавать приложения для Windows 8. Полную версию статьи вы можете найти на странице http://www.informit.com/articles/article.aspx?p=1853667.

Если не вдаваться в детали, вот основные причины, которые, на мой взгляд, позволят вам получать удовольствие от работы на новой платформе:

- **Поддержка разных языков программирования.** Приложения для Windows 8 можно писать на VB, C#, C++ и XAML или использовать набор технологий, который включает в себя HTML5 и JavaScript.
- **XAML.** Разработчики, которые знакомы с мощью и гибкостью XAML и которые раньше занимались созданием приложений для Silverlignt или WPF, будут уверенно чувствовать себя при работе над приложениями для Windows 8 с использованием XAML.
- **HTML5.** Широкая поддержка HTML5 в качестве одного из языков описания разметки привлекательна для тех веб-программистов, кто переходит на программирование для планшетных компьютеров, где основной упор делается на сенсорное взаимодействие с пользователем. Хотя эта книга посвящена преимущественно работе с C# и XAML.
- **Среда выполнения Windows (WinRT).** WinRT содержит множество элементов управления, компонентов, классов и методов, которые позволяют решать сложные задачи с помощью всего нескольких строк кода.
- **Контракты.** Система «контрактов» представляет новый уровень организации общего доступа к данным и взаимодействия между приложением и пользователем.

- **Поддержка асинхронных операций.** Поддержка команд `await` и `async` сделала разработку многопоточного кода простой как никогда.
- **Сенсорное взаимодействие.** Сенсорному вводу данных в Windows 8 уделяется особое внимание. При обработке сенсорных событий и манипуляций такой ввод по умолчанию поддерживают все доступные элементы управления и API.
- **Параметры.** Благодаря заданию параметров посредством контрактов разработчики получают возможность предоставить конечному пользователю весьма последовательный и привычный механизм настройки приложения.
- **Перемещаемые профили.** Создавать код, который через облако синхронизирует состояние Windows 8 на разных компьютерах, стало легко и просто. (Вы можете предоставить общий доступ к файлу с помощью буквально одной строчки кода.)
- **Значки.** Windows 8 представляет набор заранее подготовленных значков, которые вы можете использовать для создания единообразных командных интерфейсов приложения.

Для того чтобы избежать путаницы, в этой книге я упоминаю программы нового вида, созданные специально для Windows 8, как «приложения для Windows 8». Шаблоны Visual Studio 2012, предназначенные для создания новых приложений, собраны в группу с названием «Магазин Windows» (Windows Store). Несмотря на то что эти приложения могут распространяться с помощью Магазина Windows, вы можете распространять через Магазин и классические настольные приложения. Поэтому я использую термин «Магазин Windows» лишь тогда, когда упоминаю шаблоны Visual Studio 2012 или когда сравниваю приложения нового типа с традиционными настольными приложениями. Во всех остальных случаях вы встретите упоминания об этих приложениях как о «приложениях для Windows 8».

Указанные десять причин — это лишь поверхностный обзор новой платформы. Windows 8 отличается от предыдущих выпусков Windows, поэтому и воспринимать эту операционную систему следует по-другому. Вам понадобится приспособиться к новому интерфейсу, который, хотя и ориентирован, в первую очередь, на сенсорное взаимодействие с пользователем, поддерживает также возможность применения мыши и клавиатуры. Вы сможете задействовать код, который прозрачно вызывает системные неуправляемые компоненты, сможете работать с новым набором элементов управления и другими объектами. Основное назначение книги, которую вы держите в руках, — быстро провести вас по пути освоения новой области знаний, чтобы вы смогли начать создавать замечательные современные приложения, используя свои прежние знания по C# и XAML.

О чем эта книга

Назначение книги — объяснить, как писать приложения для Windows 8 с использованием технологий C#, XAML, Windows Runtime и платформы .NET. Я предполагаю, что у вас есть некоторый опыт разработки. Раскрывая базовые темы, связанные с C# и XAML, я стараюсь сосредоточиться на тех особенностях, которые характерны для разработки новых приложений. Касаясь более сложных приемов работы с C# или XAML, не связанных исключительно с платформой Windows 8, я ссылаюсь на другие книги, статьи или интернет-ресурсы, чтобы при желании вы могли продолжить изучение этих методик.

Разработчик любого уровня, как опытный, желающий перевести на новую платформу существующие приложения, так и начинающий, который хочет писать приложения для Windows 8 «с нуля», найдет в этой книге все необходимое. Она охватывает полный жизненный цикл приложения — от создания программного проекта до публикации в Магазине Windows.

Как пользоваться этой книгой

Книга призвана показать вам, как писать приложения для Windows 8 с использованием C# и XAML. Каждая глава написана так, чтобы помочь вам сначала понять фундаментальные особенности целевой платформы, а потом на базе полученных знаний создавать приложения. В книге приведены примеры текстов программ, которые демонстрируют возможности системы, и рекомендации по работе с ними. Многие главы построены на основе предыдущих. Это позволяет постепенно рассказать обо всех компонентах, из которых состоит типичное приложение для Windows 8.

Все главы устроены схожим образом. Они начинаются с введения в тему и обзора возможностей, доступных в рамках рассматриваемой темы. Далее следуют примеры кода и пошаговые руководства, показывающие применение описываемой технологии. Примеры кода подробно разъясняются, завершается глава подведением итогов, чтобы вы обратили внимание на самые важные моменты.

Я рассчитываю, что вы прочитаете эту книгу от начала до конца, независимо от тех знаний и навыков, которыми обладаете. При чтении каждой главы вы обнаружите, что начинаете понимать материал все лучше и лучше, а новые понятия будут усиливать друг друга и связываться воедино. После того как вы полностью прочтете книгу, вы сможете использовать ее как справочник, обращаясь к отдельным главам в любое время, когда вам потребуется прояснить конкретный вопрос.

Мой опыт работы с технологиями Microsoft

Моя первая программа была написана на языке BASIC для компьютера TI-99/4A. Потом я программировал на ассемблере для Commodore 64, изучал C и C++ на системах, основанных на Unix. Позднее я создавал программное обеспечение для управления цепочками поставок на компьютере среднего класса AS/400 (теперь известном как iSeries). Последние 20 лет я занимался в основном разработкой масштабируемых, обладающих высоким уровнем параллелизма корпоративных веб-приложений.

Я начал работать с Silverlight сразу перед выходом версии 3.0. В то время я руководил работой команды из 12 разработчиков, которые занимались мобильной платформой менеджмента на основе ASP .NET. Эта платформа интенсивно использовала AJAX, благодаря чему мобильные пользователи получали возможности, близкие к функционалу настольных приложений. Когда стало понятно, что команда тратит больше времени на изучение веб-технологий и тестирование приложений, а не на разработку основной функциональности, я начал искать альтернативу и понял, что Silverlight — это то, что нужно.

После того как состоялся переход на новую платформу, я работал над корпоративными XAML-приложениями, а также над веб-приложениями, обладающими высокой степенью масштабируемости и построенными на основе ASP.NET MVC. В дополнение к работе над мобильным программным обеспечением для менеджмента я помогал создавать систему мониторинга серверов, которые поддерживали потоковое видео (в режиме непосредственного вещания и по запросу) в ходе зимней олимпиады 2010 года в Ванкувере. Я работал в крупном проекте анализа социальных сетей. В этом проекте технология Silverlight обеспечивала визуализацию результатов анализа данных, целью которого было выяснение отношения пользователей к торговым маркам. Моя команда разрабатывала интерфейс торговой системы планшетных компьютеров для выездных торговых агентов. Этот интерфейс, который был интегрирован с системой оплаты, должен был дать им возможность совершать сделки. Я был членом команды, которая создала Silverlight-версию крупной платформы для чтения электронных книг. Платформа разрабатывалась с учетом требований максимальной доступности и настраиваемости, предоставляя детям средства интерактивного взаимодействия и аудиоподдержки.

Все это делалось, когда я работал в компании Wintellect, основанной хорошо известными в .NET-мире людьми Джеффри Рихтером (Jeffrey Richter), Джеффом Просисом (Jeff Prosise) и Джоном Роббинсом (John Robbins). Каждый из них написал бесчисленное количество книг о технологиях Microsoft, .NET Framework и Core Language Runtime (CLR). Они обучили

тысячи сотрудников Microsoft (иногда в Microsoft требуют прохождения этих курсов перед началом работы над своими проектами) и внесли вклад в саму среду исполнения, спроектировав и написав некоторые ее части. Благодаря компании, я мог общаться с ведущими специалистами и архитекторами отрасли и имел доступ к их рекомендациям и решениям по созданию успешных корпоративных приложений.

У меня есть сертификаты по различным XAML-технологиям, в том числе сертификат разработчика Microsoft Silverlight (MCTS) и WPF (MCP). Мне присудили звание MVP в области Silverlight в июле 2010-го и повторно присваивали это звание в 2011 и 2012 годах. Преимущественно благодаря тому, что я веду блог, посвященный XAML, пишу об этом в Твиттере, выступаю с соответствующими докладами в группах пользователей и на конференциях по всей стране. Я провел практические занятия и тренинги по Windows 8, работая с ранними выпусками системы, и продолжаю вести блог об этой платформе и писать о ней. Опыт работы с XAML и понимание того, как создавать серверные и веб-приложения, дали мне ценные идеи в области разработки приложений для Windows 8.

От издательства

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

Мы будем рады узнать ваше мнение!

Все исходные тексты, приведенные в книге, вы можете найти по адресу http://www.piter.com.

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

Благодарности

Я понял, что технические книги, вроде этой, могут считаться написанными командой, даже если на их обложке указан лишь один автор. Джоан Мюррей (Joan Murray) снова помогла мне довести работу над книгой до конца, поддерживала и вдохновляла меня. Она — член великолепной команды. Особую благодарность я хочу выразить редактору-консультанту Элли Бру (Ellie Bru) за то, что она всегда была в курсе дел по каждому черновику, каждому рисунку, каждой правке, каждому отзыву. Спасибо Лори Лайонс (Lori Lyons) и Кристалл Уайт (Christal White) за то, что помогали последовательно излагать материал и исправляли мой текст, давая мне возможность казаться классным писателем.

Я испытываю огромную признательность к моим сослуживцам в Wintellect за их поддержку. Стив Портер (Steve Porter) и Тодд Файн (Todd Fine) опять помогали мне находить баланс между писательством (по утрам и вечерам) и основной работой (днем). Джеффри Рихтер (Jeffrey Richter) и Джефф Просис (Jeff Prosise) дали мне массу ценных сведений и мудрых советов, основанных на их многолетнем опыте написания отличных книг. Примеры кода от Джеффри и лабораторные работы Джеффа были бесценными инструментами, которые помогли мне изучить новую платформу. Соответствующие сценарии я привожу в этой книге. Джон Гарланд (John Garland) был моим спутником в изучении новой платформы и новых технологий. Он поддержал меня и как прекрасный технический редактор, придав нужный формат тексту книги и структурировав его.

Особое спасибо компании Telerik за разностороннюю поддержку. Я выражаю признательность Джессу Либерти (Jesse Liberty) за то, что он привлекал меня к своим подкастам, Крису Селлсу (Chris Sells) за его помощь в области HTML и JavaScript и Майклу Крампу (Michael Crump) за то, что он

не только поддерживал книгу на разных уровнях, но и нашел время сделать ценные замечания в качестве технического редактора.

Благодарю команду Microsoft, которая была со мной во время разработки системы и появления ее новых выпусков: от редакции Developer Preview до Consumer Preview, и т. д. Спасибо Джейми Родригес (Jaime Rodriguez), Тиму Хейеру (Tim Heuer), Джоанне Мэнсон (Joanna Mason), Дженнифер Марсман (Jennifer Marsman) и Лайле Дрисколл (Layla Driscoll) за их знания и проницательность. Дэвид Кен (David Kean), я выражаю вам признательность за терпеливое разъяснение особенностей переносимых библиотек классов на саммите MVP и за последующую поддержку. Дэниэл Плайстед (Daniel Plaisted), вы оказывали мне неоценимую помощь все время.

Особую благодарность мне хочется выразить моим MVP-товарищам и тем, кто активно продвигал и поддерживал эту книгу в Интернете. Спасибо Дэвиду Зордану (Davide Zordan), Шону Вилдермуту (Shawn Wildermuth), Джеффу Альбрехту (Jeff Albrecht), Роберто Баккари (Roberto Baccari), Дэвиду Дж. Келли (David J. Kelley), Зубаиру Ахмеду (Zubair Ahmed) и Джинни Когхи (Ginny Caughey). Спасибо пользовательской группе .NET в LinkedIn (LIDNUG) и особенно — Питеру Шону (Peter Shawn) и Брайану Х. Мэдсену (Brian H. Madsen). Благодарю за поддержку и за то, что дали мне место, где я мог разделить с другими свое восхищение платформой Windows 8. Спасибо Крису Вудроффу (Chris Woodruff) и Кейт Элдер (Keith Elder) за то, что помогли мне «прожариться как следует» на их шоу.

Спасибо каждому, кто ретвитнул мой пост в Твиттере или посетил мою страничку, посвященную этой книге, в Facebook. Спасибо всем первым читателям, которые не пожалели времени на свои отзывы, и спасибо *тебе*, дорогой читатель, за то, что ты есть!

В итоге, но, конечно, не в последнюю очередь, еще раз спасибо несравненным жене и дочери за понимание, когда папа запирался в офисе поздно ночью и рано утром. Я не справился бы без вас, мои девочки!

Об авторе

Джереми Ликнесс — главный консультант в Wintellect, LLC. Он работает с корпоративными приложениями более 20 лет, 15 из которых занимается веб-приложениями на базе технологий Microsoft. Он один из первых внедрил Silverlight 3.0, работал над множеством корпоративных решений на основе Silverlight. В том числе — над системой мониторинга серверов для зимней олимпиады 2010 года в Ванкувере и над собственным продуктом Microsoft для анализа социальных сетей, который называется «Looking Glass». Он, кроме того, является консультантом и менеджером проектов в Wintellect. Джереми тесно сотрудничает с компаниями из рейтинга Fortune 500, в том числе — с Microsoft. Он уже три года подряд имеет статус MVP, был объявлен MVP года в 2010 году. Кроме того, он получил награду за вклад в развитие онлайн-сообщества Microsoft (Microsoft Community Contributor) благодаря работам в области Silverlight. Джереми — автор книги «Designing Silverlight Business Applications: Best Practices for Using Silverlight Effectively in the Enterprise» (Addison-Wesley). Он регулярно выступает с докладами, пишет статьи и ведет блог, освещая темы, интересующие сообщество разработчиков Microsoft. Его блог можно найти по адресу http://csharperimage.jeremylikness.com.

Новая среда Windows 1

Среда выполнения Windows (WinRT) — это совершенно новая платформа, которая позволяет создавать приложения для Windows 8 с использованием прикладного программного интерфейса, поддерживающего множество языков. Приложения для Магазина Windows — это полноэкранные приложения, которые рассчитаны на конкретные устройства, на сенсорное взаимодействие с пользователем и на новый пользовательский интерфейс Windows 8. Приложения для Магазина Windows называют *специализированными*, так как они ориентированы на целевое устройство. Для Windows 8 можно создавать и традиционные настольные приложения. Термином «приложения для Windows 8» в этой книге называются приложения нового типа, являющиеся разновидностью приложений для Магазина Windows, которые используют платформу WinRT. Выход в свет платформы WinRT — это одно из серьезнейших изменений в концепции Windows-разработки со времен представления .NET в конце 2000 года.

В этой главе вы совершите небольшое путешествие в прошлое, вспомнив основные среды разработки для Windows. Также вы узнаете, как растущая популярность естественных пользовательских интерфейсов (Natural User Interfaces, NUI) побудила Microsoft к энергичной реакции, которая выразилась в появлении Windows 8. Вы узнаете о приложениях для Windows 8 и о различных языках, на которых их пишут. Кроме того, я расскажу, как существующие технологии, основанные на технологии XAML, такие как WPF и Silverlight, вписываются в новую среду выполнения Windows.

Взгляд в прошлое: Win32 и .NET

> Я могу изменить свое решение быстрее, чем Билл Клинтон.
>
> *Джей Лено (Jay Leno) на презентации Windows 95, говоря о новой панели задач Windows*

В 1985 году без особого шума была выпущена первая версия Windows. Она не являлась полноценной операционной системой, а представляла собой лишь надстройку над консольной средой MS-DOS и называлась MS-DOS Executive (рис. 1.1). Десятилетие спустя, с выходом Windows 95, ситуация коренным образом изменилась. Билл Гейтс вместе с ведущим ток-шоу Джеем Лено появился на сцене перед теперь уже ставшей знаковой для Windows «Кнопкой Пуск» и показал всю мощь новой операционной системы. Она быстро завоевала популярность, оставив позади конкурентов вроде Apple.

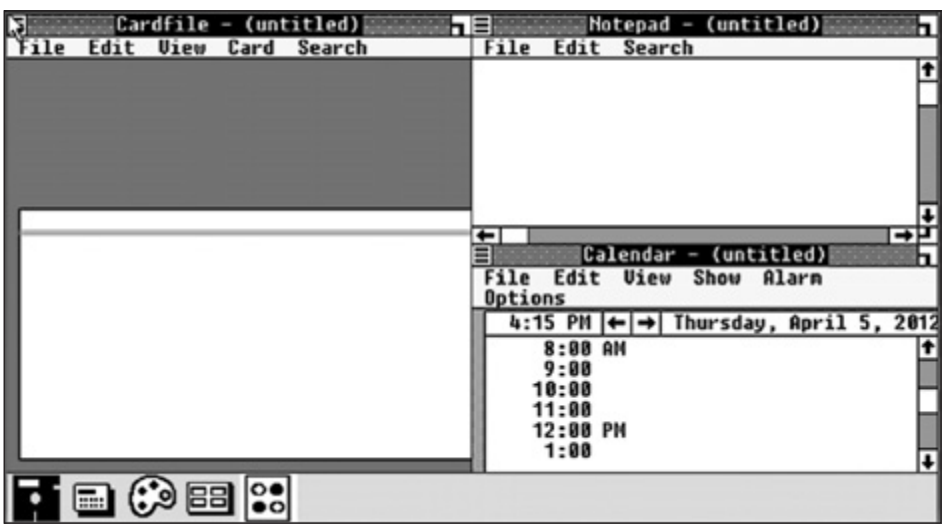


Рис. 1.1. Среда MS-DOS Executive

Для того чтобы писать программы для Windows 95 (рис. 1.2), разработчики использовали прикладной программный интерфейс (Application Programming Interface, API), созданный несколькими годами ранее и известный как Win32. В то время в Microsoft происходил переход от традиционных 16-битных систем к новым 32-битным, что и отразилось в названии

API. Win32 поддерживается и в наши дни (сегодня этот интерфейс правильнее называть Windows API) в ядре всех операционных систем Windows вне зависимости от новых сред и платформ, позволяющих от него абстрагироваться. Данный интерфейс считался весьма мощным, при этом он требовал от программиста серьезной работы по обслуживанию множества низкоуровневых операций, необходимых для вывода форм и организации взаимодействия с пользователем.

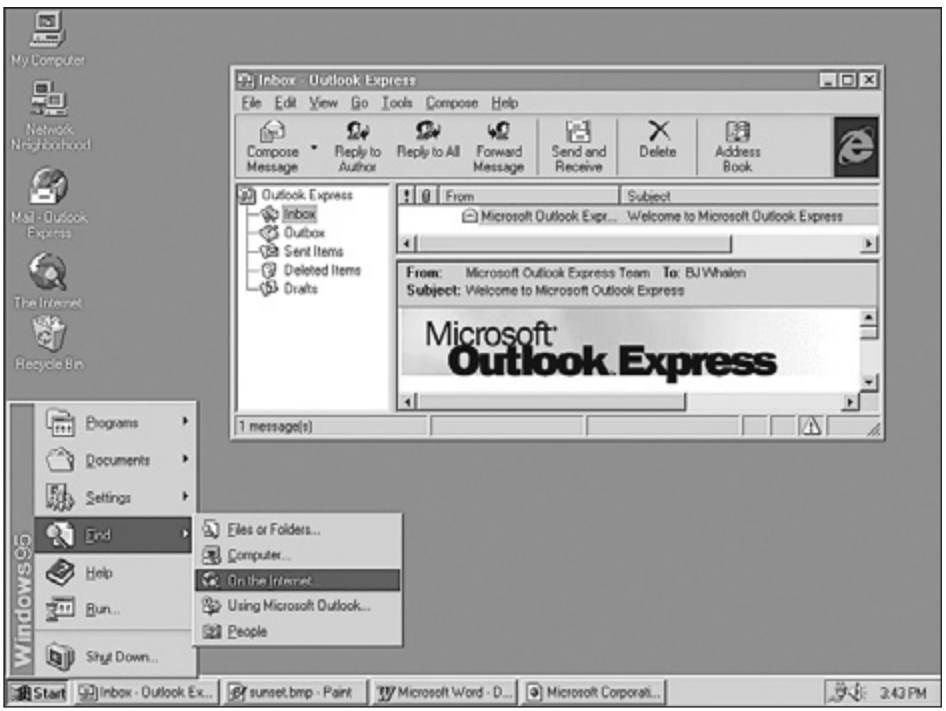


Рис. 1.2. Windows 95

Следующий код — это все, что нужно написать на языке программирования C++ для вывода текста «Hello, World»:

```
#include<iostream.h>
int main()
{
..cout << "Hello World" << endl;
..return 0;
}
```

Сравните это с кодом из листинга 1.1, который нужен для решения той же самой задачи, но с использованием API Win32. В этом примере применя-

ется библиотека базовых классов от Microsoft (Microsoft Foundation Class Library, MFC).

Листинг 1.1. Версия приложения "Hello, World", написанная с использованием Win32 и MFC

```
#include <afxwin.h>
class HelloApplication : public CWinApp
{
public:
..virtual BOOL InitInstance();
};

HelloApplication HelloApp;

class HelloWindow : public CFrameWnd
{
  CButton* m_pHelloButton;
public:
  HelloWindow();
};

BOOL HelloApplication::InitInstance()
{
  m_pMainWnd = new HelloWindow();
  m_pMainWnd->ShowWindow(m_nCmdShow);
  m_pMainWnd->UpdateWindow();
  return TRUE;
}

HelloWindow::HelloWindow()
{
  Create(NULL,
    "Hello World!",
    WS_OVERLAPPEDWINDOW|WS_HSCROLL,
    CRect(0,0,140,80));
  m_pHelloButton = new CButton();
  m_pHelloButton->Create("Hello World!",
      ↳ WS_CHILD|WS_VISIBLE,CRect(20,
    20,120,40),this,1);
}
```

Подобный прикладной программный интерфейс интенсивно использовался десятки лет при программировании для компьютеров под управлением

Windows. Этот пример иллюстрирует компромисс, который всегда был частью разработки программного обеспечения для Windows: необходимость предоставить комплексный пользовательский интерфейс с богатыми графическими возможностями требовала достаточно сложного программирования. Кроме того, многие годы это также требовало знания C или C++. Хотя существовали и другие возможности, оба языка доминировали на рынке инструментов создания того, что сейчас называется *неуправляемым кодом*. Неуправляемый код компилировался напрямую в машинные команды, которые могло прочесть целевое устройство.

Первая версия языка программирования Visual Basic (обычно его называют VB) была представлена в 1991 году. В основе VB лежали концепции, относящиеся к более старому языку, который разрабатывался специально для целей обучения и назывался BASIC (Beginner's All-purpose Symbolic Instruction Code — универсальный код символических инструкций для начинающих). Язык VB позволял разработчикам структурировать код в виде совместно работающих логических компонентов. Хотя в этом случае также генерировался машинный неуправляемый код, это происходило с помощью специальных библиотек времени выполнения, с которыми код взаимодействовал для создания готового приложения.

Язык VB 4.0, вышедший в 1995-м, позволял создавать 16- и 32-битные приложения для Windows. Интерактивная среда разработки (Interactive Development Environment, IDE), которая поставлялась вместе с VB, а также существующие компоненты и возможность создавать пользовательские интерфейсы, просто перетаскивая в нужное место элементы управления, сделали его чрезвычайно популярным среди разработчиков. К тому же, язык VB хорошо интегрировался с общей объектной моделью (Common Object Model, COM), стандартом создания программных компонентов, который был предложен Microsoft в 1993 году.

Модель COM была разработана для реализации взаимодействия между процессами и обеспечения возможности динамического создания программных компонентов (известных как объекты), к которым можно получить доступ из различных языков программирования. Модель COM позволяла разработчикам использовать компоненты, не требуя знания их внутреннего устройства. Эта возможность была реализована с помощью внешних интерфейсов. COM является важной вехой в развитии Windows, и как вы узнаете позже, даже сегодня оказывает влияние на эту платформу.

Типичная архитектура Win32-приложения показана на рис. 1.3.

Хотя API Win32 все еще существует, многие разработчики пишут отличные программы, даже не зная об этом прикладном программном интерфейсе, так как в 1990-х в Microsoft начали работу над новой средой. Ее кодовое

имя — Next Generation Windows Services (NGWS). Бета-версия этой новой среды была выпущена в конце 2000-го под общеизвестным названием .NET 1.0. Благодаря ей стало возможным создание нового типа программного обеспечения — так называемого *управляемого кода*.

Рис. 1.3. Стек компонентов Win32-приложения

При традиционном подходе к разработке программного обеспечения код назывался неуправляемым, так как он компилировался напрямую в машинные инструкции, которые могла прочитать целевая система. Хотя у такого подхода есть некоторые преимущества, разработчикам приходилось разбираться в особенностях целевой платформы на очень высоком уровне. Разработчики должны были знать, как напрямую выделять память и как ее высвобождать, когда надобность в ней отпадала. Такие задачи, как рисование графических элементов, требовали понимания механизма работы графических драйверов и того, как записывать пикселы во внутренние буферы при выводе изображения на экран компьютера.

Среда .NET Framework означала использование на платформах Windows управляемого кода. Код назвали управляемым благодаря новому слою, в котором функционировала общеязыковая среда исполнения (Common Language Runtime, CLR). Среда CLR управляла памятью вместо разработчика, предоставляла унифицированный механизм взаимодействия с различными библиотеками и ресурсами, обеспечивала создание более безопасного кода. Кроме того, CLR предоставляла независимый от языка код, в который компилировались все программы. Это код на так называемом промежуточном языке Microsoft (Microsoft Intermediate Language, MSIL), его иногда называют просто IL. Исполняемая среда интерпретировала этот код и транслировала его в машинные команды, необходимые для исполнения на целевой системе.

Концепция управляемого кода существовала задолго до .NET Framework. В 1960-х и 1970-х, когда были широко распространены мэйнфреймы, было вполне обычным написание эмуляторов для переноса программ с одной системы на другую. Компания-разработчик игр тех времен, создавшая то, что она называла «текстовой игрой» («interactive fiction»), и ставшая известной благодаря игре «Zork», была основана в 1979-м и выпускала игры для различных платформ, в том числе — для Commodore 64 и Apple IIe, используя интерпретатор «Z-machine». Разработка технологии Java началась в 1991 году, она характеризовалась использованием виртуальной Java-машины (Java Virtual Machine, JVM) для управления кодом, написанным на языке программирования Java.

Среди преимуществ, которые предоставляла эта модель, можно отметить открытость платформы для различных языков, хорошая межплатформенная поддержка, а также поддержка различных операционных систем. Программы, написанные на неуправляемом коде, часто были ориентированы на конкретные версии операционной системы или предусматривали использование специальных библиотек, созданных для того, чтобы обеспечивать их совместимость с различными операционными системами (такими как Windows XP, Windows Vista и Windows 7). В то же время управляемый код обычно ориентирован в основном на конкретную версию .NET Framework, а уже сама эта среда призвана преодолеть различия между операционными системами, на которых этот код устанавливался.

Исходная версия .NET Framework поддерживала специальный API для разработки пользовательских интерфейсов. Этот прикладной программный интерфейс, известный как Windows Forms (WinForms), был основан на интерфейсе графических устройств (Graphics Device Interface, GDI); последний предоставлял методы для непосредственного доступа к графическому аппаратному обеспечению. Среда .NET Framework 3.0 увидела свет в 2006 году и включала в себя подсистему Windows для создания презентаций (Windows Presentation Foundation, WPF). Эта новая технология была значительным шагом вперед, так как базировалась на расширяемом языке разметки приложений (Extensible Application Markup Language, XAML), применяя его в качестве языка декларативного описания пользовательских интерфейсов. XAML опирается на векторную графику и гибкие макеты, которые масштабируются под различные форм-факторы дисплеев, а также поддерживает концепцию привязки данных.

Архитектуру приложения, написанного для платформы .NET Framework, иллюстрирует рис. 1.4. В этом примере среда WPF располагается поверх среды исполнения ядра. Разработчик может задействовать различные языки и технологии, наподобие XAML, для создания кода, который

использует преимущества платформы .NET и нижележащих наборов классов. Они являются частью библиотеки базовых классов (Base Class Library, BCL) и предоставляют доступ к функциональным возможностям общего характера, таким как доступ к файловой системе и передача данных по сети. Использование любого языка приводит к созданию MSIL-кода, который с помощью CLR компилируется при исполнении в машинный код.

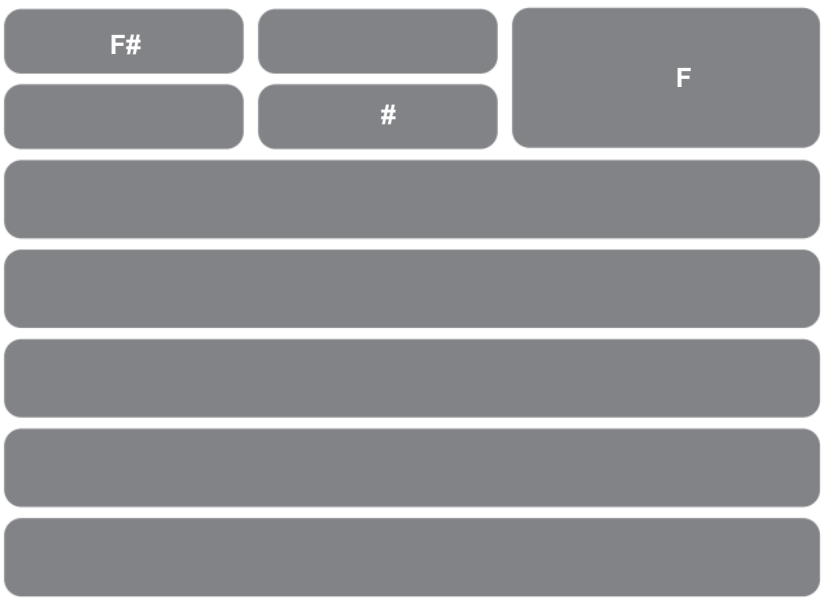


Рис. 1.4. Стек компонентов приложения, созданного с использованием .NET Framework

Технологии, основанные на .NET Framework и API Win32, играли ведущую роль в создании программного обеспечения для компьютеров, работающих под управлением Windows в течение последнего десятилетия. Хотя среды наподобие WPF привели к революционным изменениям в разработке пользовательских интерфейсов приложений для Windows, а технология Silverlight позволила охватить и другие программно-аппаратные платформы, Windows нанесли серьезный удар планшетные ПК, которые ранее не считались серьезными конкурентами традиционным системам. В апреле 2010-го, через 15 лет после презентации Windows 95, компания Apple взяла реванш, продав почти 15 миллионов планшетов iPad за первый год продаж. Этот смелый новый продукт, предоставив потребителям то, что называется естественным пользовательским интерфейсом (Natural User Interfaces, NUI), ознаменовал собой революцию.

Взгляд в будущее: становление NUI

Печатная машинка была изобретена в начале 1800-х. Первые модели оснащали клавиатурой с буквами, расположенными по алфавиту, рычаги машинок часто заедало, когда оператор быстро набирал тексты. Кристофер Шоулз (Christopher Sholes), газетный редактор, учел эту проблему в начале 1870-х, разработав раскладку клавиатуры, в которой наиболее распространенные комбинации букв были разделены. Заблуждением является предположение о том, что целью изобретения было повышение скорости набора. На самом деле целью новой раскладки было предотвращение заедания рычагов, а повышение скорости набора текстов стало следствием этого конструктивного решения (Weller, 1918).

Клавиатура выполняла механическую функцию: она подсоединялась к литерам с помощью рычагов, призванных передавать бумаге усилие удара по клавише. На ней оставался отпечаток буквы. Печатные машинки в наши дни стали раритетом, большинство клавиатур теперь электронные. Сегодняшние клавиатуры при перевозке можно сворачивать в небольшой цилиндр и подключать с помощью беспроводных интерфейсов к компьютерам, для которых они предназначены.

Несмотря на значительные технологические достижения, внешний вид клавиатур и принципы взаимодействия с ними остались практически неизменными за последнее столетие.

Изобретение мыши впервые дополнило возможности клавиатуры в конце 1960-х. Билл Инглиш (Bill English) и Дуглас Энгельбарт (Douglas Engelbart) работали на Xerox в исследовательском центре в Пало-Альто (Palo Alto Research Center, PARC), когда изобретали первые прототипы мыши (Edwards, 2008). А самая первая мышь была так названа из-за того, что подключалась к компьютеру с помощью кабеля, который напоминал хвост. Сегодня вы можете приобрести беспроводную мышь, в которой для отслеживания перемещений устройства вместо механической системы с шаром используются инфракрасные датчики или лазеры.

Вероятно, комбинация мыши и клавиатуры в наши дни — это самый распространенный инструмент взаимодействия с компьютером. Однако про него не скажешь, что он интуитивно понятен. Я полагаю, что вы — программист, если читаете эту книгу, поэтому вполне вероятно, что друзья и семья воспринимают вас как «скорую компьютерную помощь». Вполне вероятно, что вам не раз и не два приходилось ждать, пока кто-то из ваших родственников выискивал кнопки на клавиатуре, мучительно медленно вводя текст. И вы терпеливо объясняли разницу между щелчком, двойным щелчком и щелчком правой кнопкой мыши, когда тот, кому вы помогали, пытался пользоваться мышью.

Исследования различных методик взаимодействия с компьютерами начались почти тогда же, когда в начале 1970-х происходило совершенствование мыши. Идея «естественного» интерфейса сосредоточена на методе взаимодействия, который интуитивно понятен и потому легок в освоении. Если вы видели фильм «Особое мнение» с Томом Крузом, то, вероятно, с удивлением смотрели на голографические изображения, которыми он мог управлять, просто перемещая и поворачивая руки. Манипуляция объектами с помощью рук — это нечто естественное, то, чему люди учатся в весьма раннем возрасте. Поэтому людям легче шевелить руками, перемещая документ на дисплее, нежели разбираться с тем, как навести на него указатель мыши, нажать кнопку мыши, перетащить документ в нужное место и там кнопку отпустить.

Естественный пользовательский интерфейс получил известность вскоре после 2000-го года, когда большинство выпускаемых смартфонов начали оснащать сенсорными экранами. Благодаря возможности применения жестов (указание, касание, сжатие, прокрутка), упростилось взаимодействие с меню и элементами управления телефона. В 2007-м компания Apple выпустила iPhone с полностью сенсорным управлением. Интерфейс iPhone был чрезвычайно прост в освоении и использовании. Я думаю, что такой вот естественный интерфейс — одна из ключевых причин того, что устройство так быстро стало популярным. Это был первый телефон, с которым отлично справлялись обычные пользователи.

Примерно в то же время, когда устройство iPhone набирало популярность, компания Nintendo выпустила консоль Wii. Консоль комплектовалась дистанционным игровым контроллером, с помощью сенсоров которого пользователь мог взаимодействовать с консолью, перемещая и вращая контроллер. Этот интерфейс позволяет перенести привычные вам действия — наподобие бросания мяча для боулинга или взмахов клюшки для гольфа — в видеоигру, реализуя там те же движения. Консоль немедленно стала хитом и побила несколько рекордов по объемам продаж. В частности, она стала самой продаваемой консолью месяца в США (http://en.wikipedia.org/wiki/Wii).

В 2009-м появились слухи о проекте Microsoft с кодовым именем «Project Natal». Осенью 2010 года Project Natal был выпущен в виде продукта Kinect для Xbox 360, он расширил горизонты концепции, примененной в Wii, избавившись от необходимости применения дистанционного контроллера. Для анализа объектов в помещении сенсор Kinect использует специальные камеры, которые воспринимают изображение и глубину пространства. Игроки тоже являются объектами. Камеры позволяют взаимодействовать с устройством посредством движений тела без какого-либо игрового контроллера. Массив микрофонов, расположенных на устройстве, позволяет

точно детектировать звуки. В итоге устройству можно давать голосовые команды из любой части помещения.

Популярность Kinect привела к развитию нескольких проектов с открытым кодом, направленных на создание драйверов и программного обеспечения для компьютеров. После появления нескольких конкурирующих (и неофициальных) продуктов компания Microsoft выпустила официальный пакет Kinect SDK для Windows в июне 2012-го. Его версия 1.5 включает в себя драйверы, совместимые с Windows 8.

Революционное продвижение NUI продолжалось в течение последних нескольких лет. Выросли продажи различных видов планшетных ПК. Быстрое внедрение этих устройств породило явление, которое называют «Ориентирование ИТ на потребителя» («Consumerization of IT»)[1]. Сотрудники компаний отказываются таскать огромные рабочие ноутбуки, предпочитая им более легкие, простые в использовании планшетные компьютеры с сенсорным управлением, которые они готовы приобрести за собственные деньги и принести из дома. ИТ-подразделения отреагировали на эту тенденцию, включив устройства этого типа в состав оборудования рабочего места.

Операционная система Windows 7 (рис. 1.5) имеет API для поддержания сенсорного взаимодействия с устройством. Этот прикладной программный интерфейс позволяет разработчикам создавать программное обеспечение, способное реагировать на жесты и касания, выполняемые на сенсорном дисплее. Проблема заключается в том, что большая часть программного обеспечения, написанного для данной платформы, все еще ориентировано на применение классической комбинации клавиатуры и мыши. Многие из этих программ реагируют на касания пальцев, воспринимая их как щелчки мышью. Из-за этого с такими программами неудобно работать подобным образом, так как они не предоставляют подходящей рабочей поверхности для управления контентом. Пользователи, имеющие «большие пальцы» (в сравнении с размером пера или ручки для сенсорного ввода данных), находят практически невозможным выделить текст или нажать кнопку-переключатель, поскольку те занимают слишком маленькие области на экране.

Я думаю, что когда пользователи поняли, как легко взаимодействовать с устройствами, реализующими NUI, они начали отходить от традиционных моделей управления, требующих наличия мыши и клавиатуры. В Microsoft отреагировали на это с помощью пакета Kinect в секторе развлечений

[1] Одна из крупнейших исследовательских компаний Gartner в апреле 2007 года выпустила доклад, посвященный ориентированию ИТ на потребителя. Вы можете прочитать общие выводы этого доклада по адресу http://www.gartner.com/DisplayDocument?id=503272.

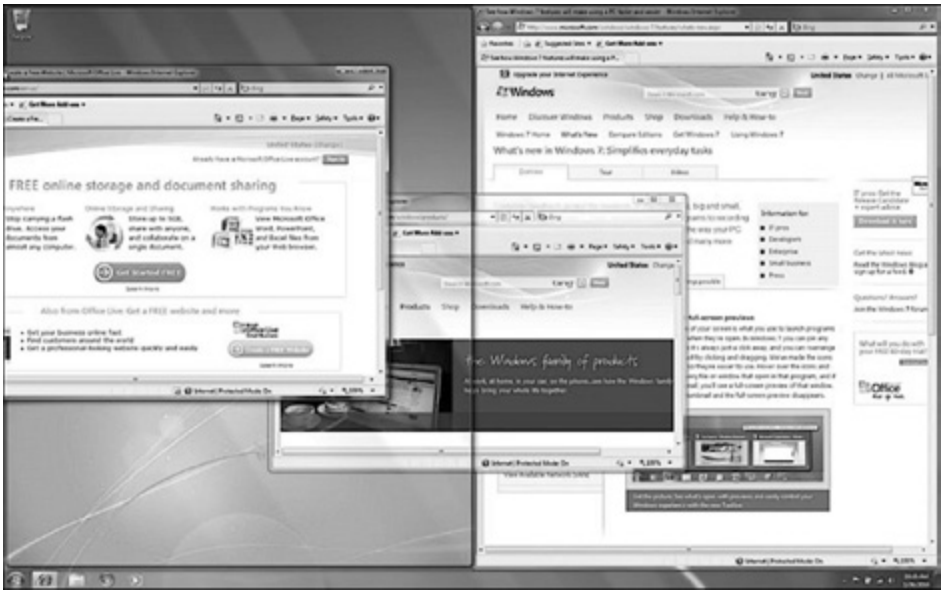


Рис. 1.5. Windows 7

и с помощью новой операционной системы Windows Phone в секторе смартфонов. Компания поняла, что ей нужно что-то соответствующее духу времени в отношении компьютеров под управлением Windows, что-то, что подразумевает не только сенсорный интерфейс. Этим «что-то» стала операционная система, оптимизированная для сенсорного взаимодействия с обычными пользователями. Она была призвана дать пользователю возможность просто взять планшетный компьютер и немедленно приступить к работе с ним на полную мощность. Причем, в отличие от Apple и Android, Microsoft нужно было делать все это, не имея такого преимущества, как возможность создания совершенно новой платформы. В мире по состоянию на декабрь 2011-го существовало 1,25 миллиарда компьютеров под управлением Windows (из них 500 миллионов — под управлением Windows 7), на которых функционировало программное обеспечение, написанное за несколько последних десятилетий. Об этом программном обеспечении нельзя было забывать, нельзя было оставить его без внимания при выпуске новой версии операционной системы (http://articles.businessinsider.com/2011-12-06/tech/30481049_1_android-apps-ios).

Ответом Microsoft на этот вызов стала революционная операционная система Windows 8, построенная на привычной архитектуре Windows и позволяющая пользователям запускать те же приложения, с которыми они работают на компьютерах под управлением Windows 7, Windows Vista, и Windows XP. Платформа преобразилась, предложив качественный сен-

сорный естественный пользовательский интерфейс. Это стало возможным благодаря приложениям нового типа: специализированным приложениям для Магазина Windows.

Знакомство с приложениями для Магазина Windows

Операционная система Windows 8 ориентирована на решение различных задач. Здесь и обеспечение обратной совместимости для большой клиентской базы, и реализация естественного пользовательского интерфейса, и учет форм-фактора планшетных компьютеров. Ключевая особенность Windows 8 — специальный тип приложений, написанных для новой среды исполнения Windows (WinRT). Хотя эти приложения обычно называют приложениями для Магазина Windows, здесь и далее в этой книге мы будем называть их приложениями для Windows 8.

Приложения для Windows 8 специально ориентированы на пользователя, который их запускает. В них учтены специальные возможности аппаратного обеспечения, к тому же они могут адаптироваться к контексту, в котором исполняются. Они масштабируются в соответствии с множеством вариантов разрешения и ориентации экрана. Эти приложения без проблем поддерживают управление с помощью мыши и клавиатуры, когда недоступны возможности сенсорного дисплея. Они могут опираться на показания датчиков при определении местоположения пользователя и реагируют на перемещения устройства, на котором выполняются.

Приложения для Windows 8 должны быть активными, быстрыми и гибкими, они должны легко взаимодействовать с другими программами, с социальными сетями и облачными сервисами, они должны быть простыми в освоении и интуитивно понятными в использовании, так как опираются на естественные жесты и движения. Опыт взаимодействия с приложениями для Windows 8 распространяется и на цикл их разработки. С помощью предоставляемых средств разработки процесс создания качественных приложений на выбранном вами языке должен быть быстрым и простым.

Платформа приложений для Windows 8 исполняется с помощью специального уровня абстракции, который называется средой выполнения Windows (Windows Runtime, или WinRT). С использованием техники, которая называется *проекцией*, WinRT отображает системные прикладные программные интерфейсы на объекты выбранного вами языка программирования. В C# взаимодействие с ними выглядит и воспринимается как взаимодействие классов. Эта техника упрощает работу без снижения производительности.

Дело в том, что скомпилированный код вызывает API напрямую, как если бы вы писали программу на низкоуровневом неуправляемом языке C или C++. Здесь не применяется промежуточная трансляция или отображение, API транслируется напрямую в ОС Windows 8, получая непосредственный доступ к возможностям операционной системы. В отличие от API Win32, WinRT — это объектно-ориентированный прикладной программный интерфейс.

API WinRT предоставляется с помощью той же техники, что и в среде .NET. Общеязыковая инфраструктура (Common Language Infrastructure, CLI) призвана снабжать метаданными о прикладных программных интерфейсах, которые компилятор может использовать для проецирования их сигнатур в выбранный язык и для компиляции в машинный код. Метаданные соответствуют стандарту ECMA-335 и хранятся в файлах с расширением .winmd (Windows Metadata — *метаданные Windows*). Дополнительную информацию об этом стандарте вы можете найти на веб-сайте ECMA: http://www.ecma-international.org/publications/standards/Ecma-335.htm.

На рис. 1.6 показана архитектура, которой вы будете пользоваться при создании приложений для Windows 8. Существует большое количество прикладных программных интерфейсов, которые поставляются вместе с Windows 8 и служат для решения самых разных задач: вывод графики, взаимодействие с устройствами, обеспечение безопасности, сетевой обмен данными, взаимодействие с операционной системой, обмен информацией с другими приложениями. В настоящий момент платформа разработки приложений для Магазина Windows поддерживает четыре языка и две графические подсистемы описания разметки. Одна из них основана на XAML (она поддерживается и из неуправляемого кода, и посредством

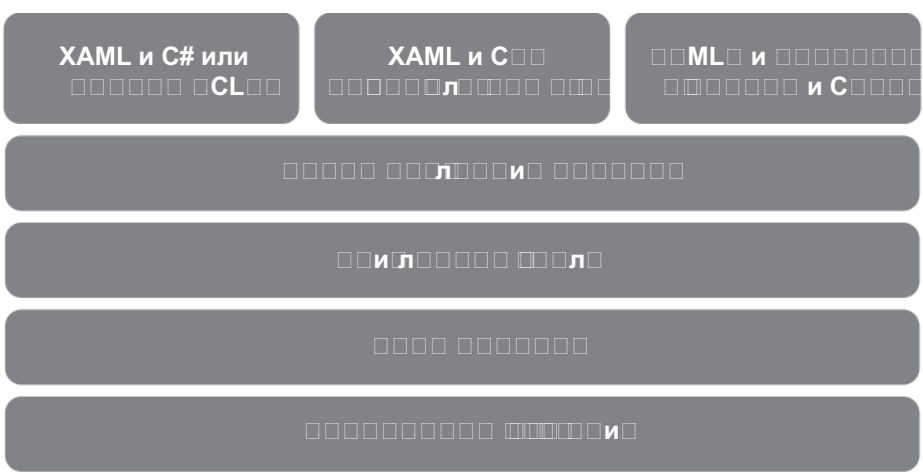


Рис. 1.6. Стек компонентов приложений для Windows 8

CLR), вторая основана на HTML5 (использует внутреннюю подсистему «Trident» для визуализации данных и встроенную в Internet Explorer 10 подсистему «Chakra» для интерпретации JavaScript-кода).

Важно, чтобы вы были знакомы с особенностями приложений для Windows 8. Основной графический интерфейс для этих приложений — DirectX, в них не предусмотрен непосредственный доступ к устаревшему интерфейсу графических устройств (Graphics Device Interface, GDI). У приложений для Windows 8 не бывает перекрывающихся окон. Они исполняются в специальном «прикладном контейнере» («Application Container»), который определяет различные состояния приложения. Приложение для Windows 8 может быть приостановлено (suspended), когда оно не активно, или остановлено (terminated) в случае нехватки системных ресурсов, таких как оперативная память.

В API WinRT применяются два варианта действий: непосредственные вызовы нижележащих функций ядра и вызовы API через брокера. Вызовы API через брокера — это особые вызовы, которые могут влиять на целостность и безопасность данных, на сохранность информации пользователя. Для того чтобы использовать эти специальные прикладные программные интерфейсы, приложение для Windows 8 должно «заявить о намерении». Для этого в манифесте приложения указываются возможности, соответствующие предполагаемым вызовам. Пользователю обычно задают вопрос о предоставлении разрешения (то есть требуется его явное согласие) на выполнение подобных вызовов еще до того, как приложению будет позволено это сделать.

Дженсен Харрис (Jensen Harris), один из руководящих сотрудников Microsoft, представил на конференции Microsoft Build доклад, посвященный восьми характерным признакам хорошо написанных приложений для Windows 8[1]. В Microsoft тщательно скрывали подробности о Windows 8 до конференции Build, на которой была формально представлена операционная система и выпущена тестовая версия для разработчиков программного обеспечения, доступная широкой публике. Помимо упомянутых признаков Дженсен в докладе дал рекомендации, на которые следует ориентироваться при создании приложений для Windows 8.

Дизайн Windows 8

Приложения для Windows 8 обладают единообразным внешним видом и поведением. Для того чтобы упростить интуитивное освоение приложе-

[1] Вы можете найти его замечательную презентацию по адресу http://channel9.msdn.com/Events/BUILD/BUILD2011/BPS-1004.

ния пользователем, вам нужно следовать руководствам по разработке приложений для Windows 8 настолько точно, насколько это возможно. Вместе со средой разработки поставляется несколько шаблонов, что упрощает выполнение данной рекомендации. Из этой книги вы узнаете о различных руководствах и рекомендованных приемах.

Вы можете прочитать руководства Microsoft по разработке приложений для Windows 8 по адресу http://msdn.microsoft.com/ru-ru/library/windows/apps/hh465427.

Скорость и гибкость

Все приложения для Windows 8 должны быть быстрыми и гибкими. Платформа помогает управлять скоростью отклика приложений, позволяя осуществлять асинхронный доступ к прикладным программным интерфейсам, вызовы которых могут выполняться медленно. В случае с приложениями для Windows 8 термин «медленно» применим к любой операции, выполнение которой может потребовать более чем 50 миллисекунд. В качестве примеров потенциально медленных операций можно привести доступ к файловой системе, передачу данных по сети. Асинхронные операции позволяют избежать блокирования пользовательского интерфейса в то время, когда операция выполняется в фоновом режиме. Некоторые усовершенствования языка C# и нижележащей платформы упрощают управление асинхронными вызовами, ожидание их результата и реагирование на этот результат в коде приложения.

Шаблоны Visual Studio 2012 и интегрированная среда разработки (IDE) содержат встроенную анимацию для создания гибкого пользовательского интерфейса. Большинство приложений поддерживают прокрутку в горизонтальной плоскости и возможность контекстного масштабирования для показа дополнительного контента или более подробных сведений о нем. В главе 2 вы узнаете, как различные шаблоны обеспечивают возможность гибких анимированных переходов. А о том, как применить более совершенные переходы, рассказывается в главе 3.

Фиксация и масштабирование

Приложения для Windows 8 можно легко зафиксировать в нужном месте экрана, чтобы два приложения делили экран и выполнялись рядом друг с другом, если разрешение дисплея достаточно велико (требуется, как минимум, 1344 пиксела в ширину). Для этого пользователь может сделать

определенный жест, управляя запущенными приложениями. Приложения, кроме того, легко масштабируются в соответствии с имеющимся пространством. Они увеличивают или уменьшают окна, когда доступное пространство становится больше или меньше или пользователь поворачивает планшетный компьютер, меняя его ориентацию с портретной на альбомную и наоборот. Встроенные шаблоны в Visual Studio обеспечивают реализацию этих возможностей, на что можно опираться при создании собственных приложений.

Использование подходящих контрактов

В приложениях для Windows 8 и WinRT предложена новая концепция, которая называется *контрактом*. Воспринимайте **контракты** как независимое от языка средство выражения в коде обязательств по выполнению определенных действий. Контракт **совместного доступа** (Share), например, похож на мощный буфер обмена, так как он поддерживает множество типов данных, включая HTML-контент и растровые изображения. Контракты позволяют разработчику предоставлять доступ к службам и взаимодействовать с операционной системой напрямую. Также они могут быть активированы пользователем с помощью панели чудо-кнопок (charms). Чудо-кнопки — это одна из функциональных возможностей системного пользовательского интерфейса и часть платформы Windows 8, которая дает пользователю возможность активировать контакты посредством единообразного пользовательского интерфейса. Еще один пример контракта и соответствующей чудо-кнопки — это контракт **поиска** (Search).

Когда пользователь нажимает чудо-кнопку Search (Поиск), находясь в приложении для Windows 8, система показывает ему панель с текстовым полем, в которое он может ввести поисковый запрос. Ниже текстового поля операционная система выведет все программы, которые поддерживают контракт поиска. Операционная система передаст поисковый запрос любой выбранной программе, таким образом, приложение сможет обработать запрос в соответствующем контексте. Приложение для работы с видео может выполнить поиск среди заголовков видеоклипов; приложение для Windows 8, которое агрегирует RSS-каналы, может осуществить поиск по текстам свежих новостей. Этот процесс иллюстрирует рис. 1.7.

Другие чудо-кнопки: Settings (Параметры), Devices (Устройства) и Share (Общий доступ). Последняя служит для обмена данными между приложениями. Больше о чудо-кнопках и контрактах вы узнаете в главе 8.

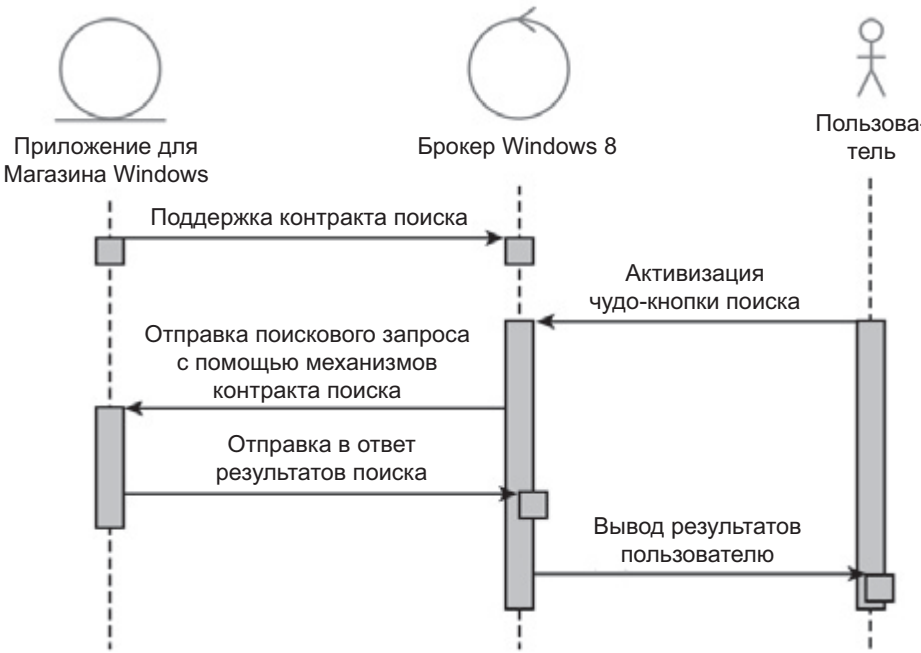


Рис. 1.7. Чудо-кнопки и контракты

Эти великолепные плитки

Впервые концепция плиток была представлена Microsoft в Windows Phone 7. В отличие от значков, которые занимают место на рабочем столе исключительно для того, чтобы щелкать на них для запуска приложений, плитки активны. Это интерактивное пространство, способное предоставлять пользователю динамически изменяющуюся информацию и контекстные сведения. Плитка погодного приложения будет показывать сведения о текущей температуре воздуха и краткий прогноз погоды. Плитка приложения для Твиттера может циклически воспроизводить свежие записи, в которых упоминается ваша учетная запись. Плитка почтового клиента покажет текущее количество непрочитанных сообщений.

Благодаря активным плиткам начальный экран Windows 8 трансформировал меню кнопки Пуск (Start) предыдущих версий этой ОС в полнофункциональную панель, богатая информация которой доступна «с одного взгляда». Часто можно получить нужную информацию, даже не запуская приложение, которому соответствует плитка. Это напоминает механизмы

взаимодействия ранних версий Windows, поддерживаемые технологиями Active Desktop (http://ru.wikipedia.org/wiki/Active_Desktop) и гаджетов рабочего стола (http://msdn.microsoft.com/en-us/library/windows/desktop/dd834142.aspx). Однако плитки являются неотъемлемой частью Windows 8 и напрямую связаны с вашим приложением.

Вы можете увидеть пример прежнего статичного рабочего стола на рис. 1.8. Единственный «активный» фрагмент данных — дата и время в его левой нижней части. Все остальное статично, значки служат лишь ярлыками для запуска приложений. Обратите внимание на большое количество «пустого пространства».

Рис. 1.8. Статичный рабочий стол

Теперь взгляните на рис. 1.9. Это начальный экран Windows 8. Обратите внимание на то, как используется его пространство. Погодное приложение показывает сведения о текущей погоде, и для того чтобы их увидеть, достаточно взглянуть на плитку (да, в Вудстоке, штат Джорджия, сейчас отличный денек). Приложение для работы с фотографиями циклически показывает свежие фотографии. А в правой части экрана я предпочел

использовать маленькие плитки и выключить режим активного обновления. Почти все в Windows 8 поддается настройке.

Рис. 1.9. Начальный экран Windows 8

Инфраструктура обновления плиток разработана с учетом экономии ресурсов, таких как объем передаваемых по сети данных и продолжительность работы устройства от батарей. Больше о плитках вы узнаете в главе 7.

Подключены и активны

Приложения для Windows 8 должны быть всегда активными и всегда оставаться на связи. Это подразумевает, что они постоянно предоставляют вам свежую информацию посредством плиток, уведомлений и собственного контента. С помощью контрактов и чудо-кнопок они могут легко подключаться к социальным сетям, облачным хранилищам данных и локальным устройствам. В дополнение к чудо-кнопкам и контрактам, это достигается путем интеграции данных из различных источников (больше о работе с данными вы можете узнать в главе 6) и трансляции богатого сетевого контента.

Реализация принципов разработки Windows 8

Последний признак хорошего приложения для Windows 8 заключается в том, что оно должно соответствовать принципам разработки Windows 8. Этот признак, на самом деле, подразумевает все остальные. Когда вы разрабатываете приложение, сделайте его быстрым и реактивным. Делайте

больше с меньшими усилиями. Сконцентрируйтесь на основных задачах и не добавляйте отвлекающих факторов. Уберите все, что не является необходимым, и всегда фокусируйтесь на том, что приложение должно быть удобным для конечного пользователя. Насколько быстро и просто он может решить ту или иную задачу? Как быстро он доберется до информации, которая ему нужна?

Эта книга повсеместно пропагандирует эти принципы. К счастью, в Microsoft критерии качественных приложений четко определены, и принципы их построения интегрированы в различные инструменты разработки и шаблоны. Больше о них вы узнаете из следующего раздела.

Инструменты разработки для Windows 8

Новая среда выполнения требует новых инструментов для создания приложений. Операционная система Windows 8 стала доступной для производителей с 15 августа 2012 года, а для широкой публики — с конца октября 2012 года. Среда разработки Visual Studio 2012 была выпущена в августе 2012 года. Она обеспечивает создание и развертывание новых приложений для Магазина Windows. При разработке традиционных настольных приложений их можно исполнять сразу же после компиляции. Как и приложения для Windows Phone, приложения для Windows 8, до того как вы сможете их запускать или отлаживать, сначала должны быть оформлены в пакеты и развернуты.

Visual Studio 2012 — это основной инструмент, которым вы будете пользоваться при создании приложений для Windows 8. Вы можете загрузить его со страницы http://dev.windows.com/.

Установив Visual Studio 2012, вы сможете создавать новые приложения для Windows 8 непосредственно в интегрированной среде разработки. На рис. 1.10 представлены различные варианты шаблонов, доступных при создании нового проекта. Шаблоны сгруппированы по языкам, затем следует подраздел, соответствующий приложениям для Магазина Windows (Windows Store). Существует несколько шаблонов приложений для Windows 8, которые помогут вам быстро создать основу пользовательского интерфейса приложения.

Blend для Visual Studio

Blend — это еще один инструмент разработки приложений для Windows 8. Он поддерживает как HTML-, так и XAML-приложения. В то время как

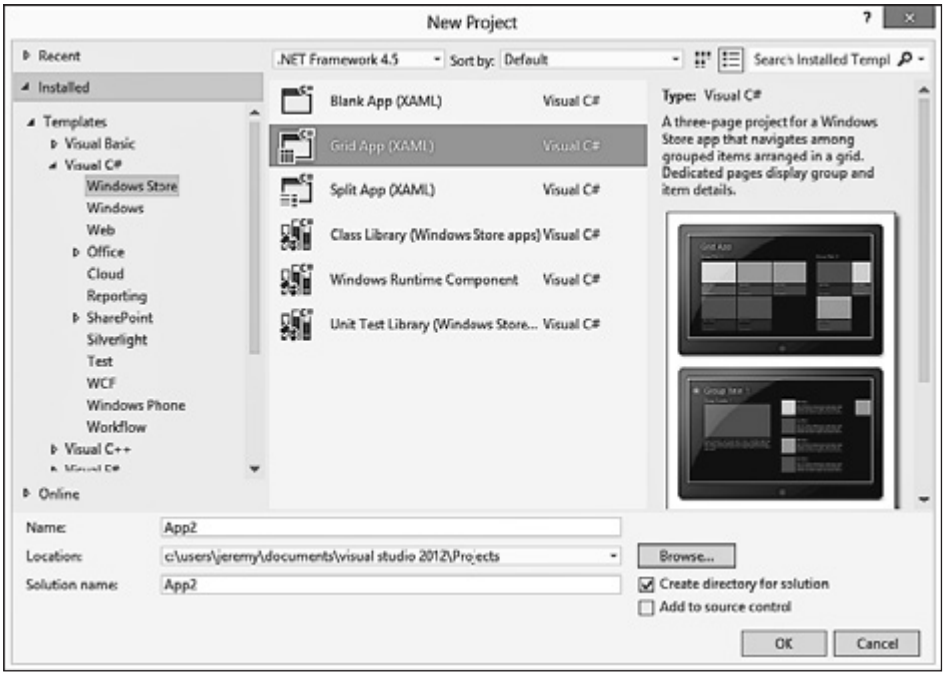


Рис. 1.10. Встроенные шаблоны приложений для Windows 8

среда Visual Studio 2012 нацелена на решение задач разработки, инструмент Blend ориентирован в основном на дизайн. В отличие от предыдущей версии Visual Studio, в которой для нужд дизайнеров использовалась отдельная подсистема визуализации, имеющая кодовое имя «Cider», ядро аналогичной подсистемы для Blend внедрено в Visual Studio 2012. Большинство действий, связанных с дизайном, можно выполнять непосредственно в среде разработки.

В Blend имеется пользовательский интерфейс для дизайна анимаций и манипулирования визуальными состояниями элемента. Об анимации вы узнаете в главе 3, а о визуальных состояниях — в главе 4. На рис. 1.11 показано окно создания нового проекта в Blend.

Хотя основное внимание в этой книге уделено тому, как путем объединения технологий C# и XAML создавать приложения для Windows 8, далее мы кратко познакомимся с другими языками и с самим языком C#.

HTML5 и JavaScript

Возможность применения JavaSctipt уникальна, поскольку подсистема разметки этого языка весьма своеобразна. То есть в создаваемом приложении

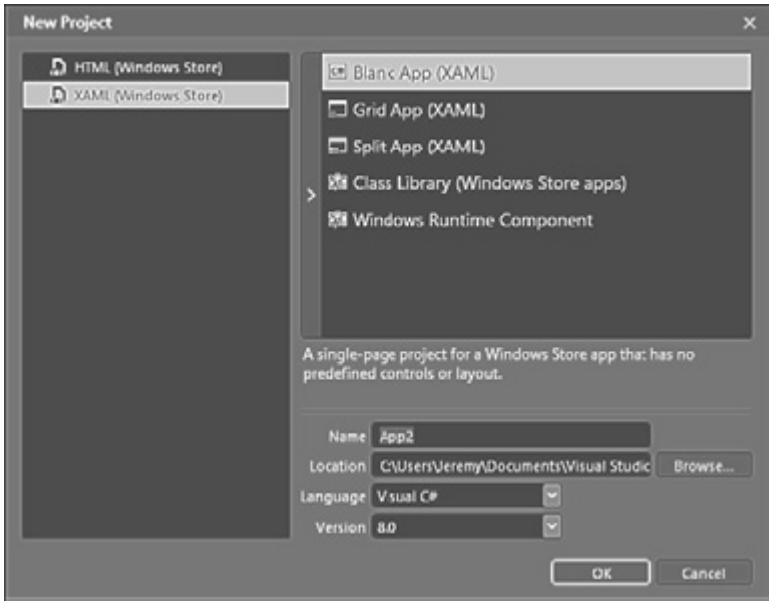


Рис. 1.11. Blend для Visual Studio 2012

для Windows 8 разметка (включая поддержку CSS) делается средствами HTML5, а программный код пишется на языке JavaScript. HTML5 использует ту же подсистему визуализации, имеющую кодовое имя «Trident», что и Internet Explorer. Для разработчика преимущество подобного подхода заключается в том, что он упрощает получение с веб-ресурса, обработку и вывод контента, основанного на HTML. Это позволяет применить знания в области HTML, CSS и JavaScript. Вы можете воспользоваться возможностями существующих инструментов, наподобие jQuery (http://jquery.com/), чтобы выбирать и модифицировать элементы страниц.

Прикладные программные интерфейсы WinRT проецируются в исполняемую среду JavaScript в виде JavaScript-объектов. С помощью полученных классов вы можете напрямую взаимодействовать с устройствами, получать данные со встроенной камеры и реагировать на состояние различных встроенных датчиков непосредственно из JavaScript-кода. Асинхронные вызовы обрабатываются в JavaScript с использованием так называемых «обязательств» (promise). Узнать больше об обязательствах вы можете на странице http://msdn.microsoft.com/ru-ru/library/windows/apps/Hh700339.aspx.

Технологии HTML/JavaScript напрямую интегрированы в Microsoft Expression Blend — мощный инструмент разработки приложений для Windows 8. Этот инструмент традиционно применялся разработчиками и дизайнерами для манипулирования пользовательским интерфейсом на базе

XAML. В Visual Studio 2012 возможности Blend расширены и позволяют дизайнерам работать также с HTML5 и CSS.

Когда следует использовать данный подход?

- Вы — опытный веб-разработчик, у вас есть знания в области HTML5, CSS и JavaScript и вы хотели бы использовать их при создании приложений для Windows 8.
- Вы работаете с командой дизайнеров, которая предпочитает технологии HTML5 и CSS.
- Вы создаете приложение, которое интенсивно взаимодействует с HTML-контентом.

Разметка для простого JavaScript-приложения «Hello, World» требует лишь нескольких строк кода, как показано в листинге 1.2.

Листинг 1.2. JavaScript-приложение "Hello, World" для Windows 8

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

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

  <!-- HelloWorldJavaScript references -->
  <link href="/css/default.css" rel="stylesheet" />
  <script src="/js/default.js"></script>
</head>
<body>
  <div id="helloText" />
</body>
</html>
```

Для того чтобы из JavaSript обновить контент элемента `helloText`, вы можете сослаться на идентификатор и установить атрибут `innerText` в файле default.js, например, так:

```
helloText.innerText = 'Hello, World.';
```

JavaScript-шаблоны предоставляют практически точно такие же возможности, как и другие. Вы обнаружите, что очень сложно определить, какой язык использовался при создании приложения, основываясь только на том, как оно выглядит и работает.

C++ и XAML

Возможность использования языка C++ весьма привлекательна для тех разработчиков, кто традиционно привык работать с неуправляемым кодом. Этот подход позволяет создавать приложения для Windows 8, применяя технологии C++ и XAML. Элементы пользовательского интерфейса, объявленные в XAML, проецируются в приложении как объекты, к ним можно получать доступ и манипулировать ими непосредственно из кода.

В XAML-коде для приложения «Hello, World», написанном на C++, просто объявляется элемент `TextBlock` для вывода текста:

```
<Grid Background="{StaticResource
ApplicationPageBackgroundThemeBrush}">
  <TextBlock x:Name="HelloText"/>
</Grid>
```

Вспомогательный код (code-behind) просто задает свойство `Text` элемента:

```
MainPage::MainPage()
{
  InitializeComponent();
  HelloText->Text = "Hello, World.";
}
```

Вы можете пользоваться аппаратно-зависимым кодом на C++ при создании игр для Windows 8, которые взаимодействуют непосредственно с DirectX. Такой подход позволяет получить доступ ко всем возможностям графического аппаратного обеспечения посредством языка высокого уровня для программирования шейдеров (High Level Shading Language, HLSL). Это требуется большинству приложений, интенсивно работающих с графикой, таких как игры. Данная возможность, кроме того, позволяет ссылаться на существующие проекты и библиотеки на C++ с целью их использования в приложениях для Windows 8.

Когда следует прибегать к данному подходу?

- Вы — опытный разработчик на C++ и желаете использовать существующие знания при создании приложений для Windows 8.

- Вы работаете с командой дизайнеров, которая хорошо знакома с XAML.
- Вы не хотите подвергать систему дополнительной нагрузке, которую вызывает использование движка веб-браузера (HTML5/JavaScript) или среды CLR (C#/XAML).
- Вы заинтересованы в написании аппаратно-зависимого кода на C++ и (или) в использовании высокопроизводительных механизмов наподобие HLSL-движка.

VB/C# и XAML

Последняя доступная возможность — это использование управляемых языков VB и C#. Уникальные свойства среды WinRT позволяют таким приложениям получать непосредственный доступ к API посредством проекций, которые выглядят как собственные CLR-объекты. Эта книга посвящена в основном C#, но вызовы API и шаблоны для версии VB.NET почти идентичны, различаются лишь языковые особенности.

XAML-код для приложения «Hello, World», написанного на C#, точно такой же, как и для приложения на C++. А вспомогательный код на C# выглядит так:

```
public MainPage()
{
  InitializeComponent();
  HelloText.Text = "Hello, World.";
}
```

Когда следует использовать данный подход?

- Вы — опытный разработчик на C# и желаете использовать существующие знания при создании приложений для Windows 8.
- Вы — Silverlight- и (или) WPF-разработчик и хотите продолжать использование XAML.
- Вы работаете с командой дизайнеров, которая знакома с XAML.

В примерах этой книги используются языки C# и XAML. Вы можете загрузить пример приложения «Hello, World» на сайте http://windows8applications.codeplex.com/.

Есть два способа получения исходного кода с помощью сервиса CodePlex. Первый заключается в щелчке по кнопке Download в верхней правой части страницы. Это позволит загрузить сжатый пакет, содержащий все самые

свежие файлы исходного кода. Второй способ — перейти на вкладку Source Code и просмотреть код непосредственно на сайте или загрузить конкретную версию кода, щелкнув на ссылке Download. Кроме того, вы можете присоединиться к проекту, щелкнув на ссылке в верхней правой части домашней страницы, чтобы получать оповещения об изменениях или исправлениях, когда они будут вноситься в материалы.

За кулисами WinRT

WinRT-компоненты — это COM-объекты. Они соответствуют правилам двоичного прикладного интерфейса (Application Binary Interface, ABI) — низкоуровневого интерфейса между компонентами, а также между операционной системой и другими компонентами. Компоненты, кроме того, реализуют специальный интерфейс `IInspectable`, который, в свою очередь, реализует интерфейс `IUnknown`.

Интерпретатор JavaScript будет использовать интерфейс `IInspecatable` для генерирования динамических языковых привязок. Интерпретатор берет имя класса, запрашивает метаданные для компонента у Windows и проецирует подходящие интерфейсы. Конечный результат заключается в том, что ваш JavaScript-код может получать доступ к WinRT-проекциям, и интерпретатор выполняет продвижение запросов к нижележащим компонентам. У интерпретатора есть собственный сборщик мусора, ответственный за очистку экземпляров объектов, когда они выходят из области видимости.

В версии .NET спроецированные классы на самом деле реализуют механизм COM Interoperability. .NET-приложение взаимодействует с CLR-классом, который работает с нижележащим WinRT-компонентом (COM), используя вызываемую оболочку времени выполнения (Runtime Callable Wrapper, RCW). Вызовы просто продвигаются нижележащим компонентам с использованием уровня проекции. Проекция управляет компонентами вместе со спроецированными типами. В итоге, опять же, собственный сборщик мусора платформы .NET заботится об объектах, выходящих из области видимости.

Говоря о комбинации C# и XAML, важно отметить то, что применяется полный вариант платформы .NET. Здесь присутствует специальный профиль .NET-приложений для Магазина Windows, который ограничивает использование основных библиотек базовых классов (BCL) небольшим набором и позволяет работать с уровнем проекции. Приложения, написанные на управляемом языке, зависят от среды исполнения платформы .NET. Дополнительная нагрузка из-за наличия проекции минимальна, поэтому

при создании приложений можно не беспокоиться о сколь-нибудь существенном снижении производительности.

При использовании C++ взаимодействие с WinRT осуществляется через набор языковых расширений. Это позволяет компилятору представлять WinRT-компоненты в виде знакомых языковых паттернов с конструкторами, деструкторами, методами и исключениями. Управление памятью в C++ обычно производится путем так называемого «подсчета ссылок», но для WinRT-компонентов все делается «закулисными» механизмами, то есть управление ссылками осуществляется автоматически.

WPF, Silverlight и синий стек

Версия 5 Silverlight была выпущена в декабре 2011 года, через несколько месяцев после того, как была официально анонсирована система Windows 8 на соответствующей конференции Build в 2011. Перед этой конференцией многие разработчики размышляли о будущем Silverlight. Ходили слухи, что и технология Silverlight, и даже .NET в целом, не будут поддерживаться новой операционной системой. Все это оказалось лишь слухами, компьютеры под управлением Windows 8 могут исполнять и полнофункциональный код среды .NET, и Silverlight-приложения как в браузере, так и вне его (Out-Of-Browser, OOB).

Есть ли место для этих приложений на новой платформе? Ответ, я считаю, безусловно положительный. В существующие приложения, реализованные на Silverlight и в среде NET с использованием WPF, вложены существенные объемы времени и ресурсов. Многие из этих программ могут выполняться на компьютерах под управлением Windows 8 в их исходном состоянии, особенно если они уже поддерживают возможности сенсорного управления. Windows 8 предлагает два различных варианта окружения, которые могут работать параллельно: это новая среда для приложений Магазина Windows, и среда классических настольных приложений. Иногда их называют «green stack» («зеленый стек»), говоря о приложениях для Магазина Windows, и «blue stack» («синий стек»), когда имеют в виду настольные приложения. Их называют так из-за цветов, которые были использованы в первых диаграммах, представленных Microsoft для описания Windows 8.

Эта книга посвящена приложениям зеленого стека, то есть приложениям для Магазина Windows. (Где бы вы ни увидели слова: «приложение для Windows 8», они означают, что я имею в виду именно этот специальный тип приложений, которые не являются настольными приложениями). Окружение настольных приложений, или синий стек, — это то, с чем

многие разработчики уже хорошо знакомы. Синий стек поддерживает платформу .NET вместе с множеством различных платформ, включая WPF и Silverlight. Вы можете продолжать писать приложения, ориентированные на данное окружение, с использованием тех языков и средств разработки, которые вам нравятся. Разработчикам в данном случае приходится принимать решение: либо создавать приложения для Windows 8 и переносить существующие приложения в данное окружение, либо продолжать поддерживать классические настольные приложения.

Некоторые приложения, возможно, не имеет смысла переносить в Магазин Windows. Хотя версия Outlook для Магазина Windows была бы весьма интересной, я не могу представить себе правку большого и сложного листа Excel в том же пользовательском интерфейсе. Также я не могу представить себе в Магазине Windows программу для разработки приложений. Именно поэтому Visual Studio 2012 работает в среде настольных приложений. Для подобных сценариев лучше подключить мышь и клавиатуру и набирать код. Мне не хотелось бы столкнуться в ближайшем будущем с корпоративными приложениями, с которыми приходилось бы работать с помощью жестов и касаний.

Многие приложения могут поддерживать оба режима работы. Основное приложение может исполняться в настольном окружении, а упрощенная версия, созданная для Магазина Windows, может предоставлять информационную панель с оповещениями и лентами новостей. Например, имеется версия Internet Explorer для Магазина Windows. А с рабочего стола вы можете запустить другую версию, имеющую традиционный настольный интерфейс и поддерживающую подключаемые модули. Нет причин ограничивать себя одной целевой средой, и тот факт, что приложения для Windows 8 поддерживают языки наподобие C#, предоставляет возможность использования ключевых библиотек кода в разных версиях приложений, ориентированных на различные целевые среды.

Одна особая версия Windows 8, которая не поддерживает синий стек, называется Windows RT. Эта версия исполняется исключительно на устройствах, построенных на базе ARM-архитектуры. ARM-устройства имеют специальный набор микросхем, оптимизированный для низкого энергопотребления и меньше нагревающийся при работе. Миниатюрные планшетные компьютеры, обычные планшетные компьютеры и смартфоны, построенные с использованием архитектуры ARM, могут иметь меньшие форм-факторы, так как им не нужен вентилятор для охлаждения главного микропроцессора. Набор машинных инструкций этого микропроцессора отличается от набора традиционных процессоров от Intel. В итоге, специальная версия Windows, созданная для ARM-устройств, поддерживает лишь приложения для Windows 8.

Выводы

Эта глава содержит краткое описание истории операционных систем семейства Windows. Также здесь описаны различные прикладные программные интерфейсы и платформы, предназначенные для создания программного обеспечения. Вы узнали о новой платформе для Магазина Windows и о связанных с ней инструментах разработки приложений для Windows 8, ориентированных на сенсорное управление. Я представил несколько разных направлений разработки таких приложений, привел обзор особенностей выбора языков программирования и разметки.

Приложения для Windows 8 имеют общие особенности, направленные на то, чтобы взаимодействие пользователя и приложения было последовательным, элегантным и исчерпывающим. Они легко интегрируются с системой и с другими приложениями. Эти приложения самостоятельно адаптируются, чтобы соответствовать целевым устройствам. Их создают с применением тех же языков и инструментов, которыми пользуются разработчики на C# и XAML. И если вы не можете дождаться того момента, когда начнете создавать приложения для этой замечательной новой платформы, не переживайте, вам достаточно просто перевернуть страницу!

Литература

Edwards, B. The Computer Mouse Turns 40. Macworld: December 9, 2008.

Weller, C.E. The Early History of the Typewriter. La Porte: Chase & Shepard, 1918.

Начало работы

2

«Hello, World» — весьма популярное приложение. Обычно это первая программа, которую пишут разработчики при изучении нового языка или платформы. В предыдущей главе вы узнали, как написать «Hello, World», используя три разных подхода. В этой главе вы узнаете, как настроить рабочую среду при разработке приложений для Windows 8. Существует много вариантов и важно выбрать тот из них, который лучше всего подойдет именно вам.

Когда рабочая среда будет настроена, я расскажу о шаблонах, которые предоставляются разработчику сразу после установки Visual Studio 2012. Эти шаблоны спроектированы для того, чтобы помочь вам начать работу над приложениями для Windows 8 и сделать эту работу настолько производительной, насколько это возможно. Они содержат встроенный код и разметку для обработки таких событий, как смена ориентации устройства и масштабирование (изменение размера) окна приложения. Вы узнаете, как с учетом назначения приложения выбрать шаблон, чтобы начать работу.

В этой главе я поэтапно рассмотрю простой пример создания приложения для Windows 8. Тем не менее это потребует реализации некоторых расширенных возможностей. На этом примере вы увидите, как просто эти возможности интегрируются в новую платформу. Вы узнаете, как происходит развертывание приложения в системе, как оно выглядит после развертывания. Думаю, вы удивитесь, когда узнаете, где в Windows 8 хранятся некоторые сведения о вашем новом приложении.

Настройка рабочей среды

При создании приложений для Windows 8 существует множество возможностей настройки рабочей среды. Для того чтобы разрабатывать

и тестировать такие приложения, нужно иметь компьютер, который работает под управлением Windows 8. Если у вас еще нет такого компьютера, существуют различные варианты установки данной операционной системы. После установки Windows 8 можете установить и настроить Visual Studio 2012 — основной инструмент разработчика приложений для Windows 8, и Blend — инструмент подготовки дизайна пользовательского интерфейса, ориентированный на технологию XAML.

Windows 8

Первый шаг, который нужно сделать перед началом разработки приложений для Windows 8, заключается в установке операционной системы. Это может показаться очевидным, но вы не сможете разрабатывать приложения для Магазина Windows 8 на других компьютерах. Вы должны использовать среду Visual Studio 2012, установленную на компьютере, работающем под управлением Windows 8. Вы можете получить дистрибутив системы различными способами, в том числе загрузив с основного веб-сайта по адресу http://dev.windows.com/.

Кроме того, вам нужно подготовить подходящий носитель информации для установки. Большинство дистрибутивов операционных систем распространяется в виде ISO-образов, где ISO расшифровывается как International Organization for Standardization (Международная организация по стандартизации). Такой образ можно использовать для создания установочного CD- или DVD-диска. Когда у вас есть образ, его можно смонтировать, воспользовавшись программой наподобие Virtual CloneDrive (http://www.slysoft.com/en/virtual-clonedrive.html). Другая возможность, если речь идет об обновлении Windows 7, заключается в использовании средства USB/DVD Download (оно совместимо с Windows 8) для подготовки DVD-диска или внешнего USB-диска (http://wudt.codeplex.com/).

После того как носитель информации будет готов к установке образа Windows 8, вам нужно выбрать конфигурацию установки. Обычно доступны следующие возможности:

- **Полная установка** (Full Install)**.** Выберите этот вариант для перезаписи данных существующей операционной системы и замены ее на Windows 8. Не забудьте создать резервную копию важных данных существующей системы, так как они могут быть утеряны при обновлении.
- **Двойная загрузка** (Dual Boot)**.** Используйте эту возможность, чтобы Windows 7 и Windows 8 работали параллельно. Когда вы включите компьютер, вам будет предложено выбрать, какую именно операционную систему вы хотите загрузить. У вас есть возможность создания

конфигурации с двойной загрузкой либо с существующего раздела на жестком диске, либо создав образ виртуального жесткого диска (Virtual Hard Disk, VHD).

- **Виртуальная машина** (Virtual Machine)**.** В данном случае создается виртуальный жесткий диск и происходит загрузка операционной системы в виртуальной машине. Есть множество доступных продуктов, реализующих виртуальную машину. Я иллюстрирую данный вариант на примере программы Oracle VirtualBox, так как она бесплатна.

Какой именно вариант вы выберете, зависит от имеющегося у вас рабочего окружения. Для написания этой книги я выбрал компьютер, целиком выделенный для Windows 8, используя вариант полной установки. Вариант с виртуальной машиной — не лучший выбор, так как Windows 8 — это новая операционная система, поэтому не все, что нужно для ее полноценной работы, реализовано в виртуальных машинах. Например, я обнаружил, что в виртуальной среде мышь работает непредсказуемо, к тому же там нет поддержки сенсорного управления, хотя у меня имеется полнофункциональный сенсорный дисплей.

WINDOWS 8 НА ВНЕШНЕМ НОСИТЕЛЕ ИНФОРМАЦИИ

Windows 8 предоставляет уникальную новую функциональную возможность, которая называется Windows to Go и позволяет установить операционную систему на внешний USB-диск. Полученный образ системы можно использовать на различных компьютерах, например на рабочем настольном ПК и домашнем ноутбуке. При первой загрузке на новом компьютере система автоматически сканирует его аппаратное обеспечение и загружает из Интернета подходящие драйверы. При следующих загрузках на этом же устройстве она применяет загруженные ранее драйверы. Хотя существуют некоторые ограничения и различия между данной версией установки Windows 8 и полной установкой (не поддерживаются процессоры на архитектуре ARM), эта возможность может показаться вам удобной в плане переносимости системы. Больше об этом вы можете узнать на странице http://technet.microsoft.com/ru-ru/library/hh831833.aspx.

Для того чтобы начать установку, продолжите чтение с раздела, соответствующего выбранной вами конфигурации. Уже все установили и готовы идти дальше? Тогда можете продолжить чтение с раздела «Привет, Windows 8» этой главы.

Полная установка

Установка Windows 8 выполняется быстро и легко. Достаточно вставить диск с дистрибутивом и следовать указаниям системы. При установке

системы вам будет предложено пройти несколько этапов настройки. Вы можете задать имя компьютера, выбрать параметры обновления системы через Интернет и задать имя пользователя для входа в Windows 8. Не забудьте о возможности использования учетной записи Microsoft (также известной как учетная запись Windows Live) для входа в систему. Это даст вам дополнительные возможности, такие как синхронизация с облаком и простая интеграция с сервисом SkyDrive.

Когда вы завершите шаги по персонализации, операционная система Windows 8 будет готова к работе.

Двойная загрузка

Возможность установки системы в конфигурации двойной загрузки позволит вам установить Windows 8 и в то же время оставить существующую ОС. Пояснения, которые будут даны далее, сделаны из предположения, что вы начинаете процесс установки под управлением Windows 7. Первый шаг заключается в подготовке пространства на жестком диске.

ВНИМАНИЕ

Следующие шаги требуют дополнительных знаний о вашей операционной системе. Они подразумевают повторное разбиение на разделы и форматирование разделов жесткого диска. Прежде чем продолжать, вам следует сделать полную резервную копию данных. Если вы не знакомы со средством Windows Disk Management или вам неудобно с ним работать, я настоятельно рекомендую выбрать другую возможность для установки системы, или работать с теми программными средствами, которые вы знаете лучше. Я вас предупредил!

Если на жестком диске нет раздела для установки Windows 8, вы можете поработать с существующим диском, сжав его, чтобы создать новый раздел. В меню кнопки Start раскройте подменю Computer и выберите команду Manage. Будет запущено приложение Computer Management. В левой навигационной панели выберите раздел Storage ▸ Disk Management. Запустится средство Windows Disk Management. Вы увидите что-то, напоминающее рис. 2.1. Щелкните правой кнопкой мыши на разделе, в которому нужно выделить свободное место, и выберите команду Shrink Volume.

Средство управления дисками просканирует том на предмет наличия свободного места. Это может занять несколько минут в зависимости от размера жесткого диска и степени его фрагментированности. Когда сканирование завершится, вы можете задать размер нового тома. Минимальный рекомендованный размер диска для Windows 8 составляет 20 Гбайт. На

рис. 2.2 показано диалоговое окно, где данный минимальный объем указан в мегабайтах.

Далее вы будете наблюдать в течение нескольких минут ваш любимый вращающийся значок. Когда работа завершится, появится новый раздел, маркированный надписью Unallocated. Его размер должен быть равен заданной

Рис. 2.1. Приложение для управления дисками в Windows

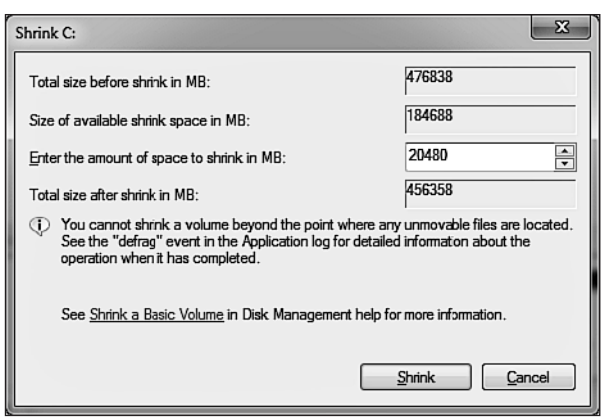


Рис. 2.2. Ввод объема дискового пространства, которое нужно освободить для создания нового раздела

величине. Щелкните правой кнопкой на новом разделе и выберите команду New Simple Volume. Появится окно мастера создания простого тома. Щелкните в этом окне на кнопке Next. Не меняйте выделенное по умолчанию место для нового тома и снова щелкните на кнопке Next. На следующем шаге снова согласитесь со значениями, предложенными по умолчанию (если только не решите назначить диску другую букву). В последнем диалоговом окне укажите, что диск следует отформатировать в файловой системе NTFS, и присвойте ему осмысленную метку. Ваше диалоговое окно должно выглядеть так, как показано на рис. 2.3.

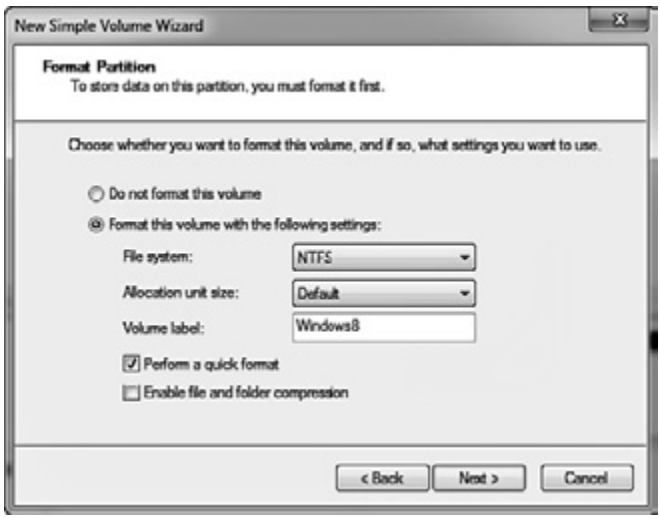


Рис. 2.3. Форматирование раздела для двойной загрузки

Щелкните на кнопке Next. Последнее диалоговое окно будет содержать перечень параметров, которые вы задали ранее. Если они верны, щелкните на кнопке Finish, и новый том будет отформатирован, смонтирован и готов к установке Windows 8.

Вставьте в дисковод DVD-диск с дистрибутивом Windows 8 или подключите USB-носитель и перезагрузитесь (инструкции, описывающие процесс установки, вы можете найти в разделе «Полная установка»). Выберите для установки вариант Custom. При указании диска для установки выберите созданный нами диск. Вы узнаете его по заданному ранее размеру и по тому, что он будет помечен как Primary. Убедитесь, что выбрали верный том; ошибка при выборе приведет к тому, что существующая операционная система будет перезаписана.

Выполните установку так, как описано в разделе «Полная установка». Когда система перезагрузится, вы увидите новое меню. В отличие от текстового меню Windows 7, загрузочное меню Windows 8 является графическим;

с его пользовательским интерфейсом можно работать при помощи мыши, клавиатуры или сенсорного дисплея. Вы увидите параметры, которые позволяют загрузить либо прежнюю ОС Windows 7, либо новую Windows 8. Очевидно, что вы выберете новую и после ее быстрой загрузки будете готовы настроить рабочую среду и приступить к созданию вашего первого приложения.

Установка на виртуальную машину

Возможно, вы решите установить Windows 8 на виртуальную машину. Последовательность действий в этом случае зависит от того, какую именно виртуальную машину вы решите использовать. Доступны несколько подобных продуктов. Одно из популярных бесплатных решений с открытым кодом называется VirtualBox. Вы можете загрузить VirtualBox с веб-сайта https://www.virtualbox.org/. Для установки ОС на виртуальную машину вам понадобится дистрибутив системы. Некоторые решения позволяют напрямую монтировать ISO-образы, чтобы упростить этот процесс.

Visual Studio 2012

Visual Studio — это предлагаемая Microsoft среда разработки программного обеспечения. Тот самый инструмент, который вы будете использовать, создавая приложения для Windows 8. Разработкой таких приложений можно заниматься, загрузив бесплатную экспресс-версию Visual Studio, которая поставляется вместе с экспресс-версией Blend. Другие версии предоставляют дополнительную функциональность, от модулей для тестирования до инструментов проектирования архитектуры приложения. В этой книге, главным образом, описываются возможности бесплатной версии, но некоторые приложения могут потребовать использования платной версии. Вот краткий список отличий между различными вариантами Visual Studio:

- **Test Professional.** Эта версия разработана для бизнес-аналитиков и тестеров. Она включает в себя средства групповой работы, множество инструментов тестирования и средства для работы с виртуальными лабораториями.
- **Professional.** Версия начального уровня для разработчиков. Она включает в себя средства тестирования и анализа кода, инструменты модульного тестирования. Данная версия поддерживает различные платформы разработки и интегрированные средства разработки.
- **Premium.** Данная версия включает в себя возможности версий Test Professional и Professional. Кроме того, она поддерживает метрики кода,

анализ покрытия кода, кодированное тестирование пользовательского интерфейса. Кроме того, она позволяет работать с UML-диаграммами, планировать спринты, управлять списком невыполненных работ по продукту, обеспечивать просмотр кода и выводить раскадровку в PowerPoint.

- **Ultimate.** Это наиболее полный выпуск Visual Studio. Он включает в себя все возможности других версий, плюс поддерживает технологию IntelliTrace (отладку по журналу), позволяет осуществлять нагрузочное тестирование, работать с Microsoft Fakes (среда изоляции модульных тестов). Кроме того, данная версия поддерживает дополнительные инструменты разработки архитектуры и предоставляет пакеты дополнительных компонентов.

Blend

Expression Blend — это инструмент дизайна, который ранее был доступен при разработке Silverlight- и WPF-приложений, а также приложений для Windows Phone. Его возможности расширены, и теперь он поддерживает приложения для Windows 8 на базе как XAML, так и HTML. Blend — это мощное средство, предлагающее современные инструменты для предварительного просмотра интерфейса приложения и для вывода данных во время разработки без необходимости запуска приложения. Кроме того, Blend поддерживает перетаскивание в область конструирования интерфейса как элементов управления, так и UI-режимов.

Многие возможности Blend интегрированы в XAML-конструктор среды Visual Studio 2012. В предыдущей версии, Visual Studio 2010, использовался XAML-конструктор под кодовым именем «Cider», который не имел таких возможностей, как Blend. Разработчики, знакомые с предыдущей версией, будут комфортно чувствовать себя в среде конструктора Visual Studio. В этой книге описываются функциональные возможности Blend, интегрированные в XAML-конструктор Visual Studio 2012.

Привет, Windows 8

В этом разделе мы создадим первое приложение для Магазина Windows. Цель создания этого приложения заключается в том, чтобы показать, как с минимальным объемом кода можно задействовать некоторые расширенные возможности системы. Приложение предоставляет возможность использовать веб-камеру, чтобы сделать снимок, который можно сохранить

в библиотеке изображений, передать в социальную сеть или в облачное хранилище данных. Кроме того, оно позволяет получать изображения от других приложений, осуществлять их предварительный просмотр и сохранять.

После того как вы создадите и запустите приложение, вы узнаете, как Windows 8 обеспечивает его работу. Вы познакомитесь с тремя фазами существования приложения для Windows 8, которые подразумевают установку, или развертывание, приложения, использование плитки в качестве средства запуска приложения и исполнение приложения. И наконец, с помощью дизассемблера ILDASM я продемонстрирую, что для представления информация вашего приложения WinRT использует те же метаданные, что и .NET.

Первое приложение для Windows 8

Все начинается в Visual Studio 2012. Здесь вы можете выбрать язык, с которым хотите работать (в нашем случае это Visual C#), и тип приложения. В зависимости от установленной версии Visual Studio вы можете создавать классические настольные приложения, веб-приложения, отчеты, SharePoint-приложения или приложения для Windows 8. Список доступных шаблонов показан на рис. 2.4.

Для того чтобы создать новый проект, выберите в меню команду File ▸ New ▸ Project. Выбрав в появившемся окне вариант Windows Store, вы увидите несколько шаблонов.

Шаблоны

Шаблоны проектов в Visual Studio предлагают различные стартовые позиции создания приложений на базе общих паттернов разработки. В зависимости от ваших нужд, вы можете решить начать с шаблона проекта, который поддерживает взаимодействие с наборами коллекций и отдельными элементами. Преимущество подобных шаблонов заключается в том, что код, который автоматически создается на их основе, поддерживает перемещение между групповым и детальным просмотром данных, реагирование на смену ориентации дисплея между портретной и альбомной, изменение размера окна приложения, когда оно фиксируется в нужной позиции на разделенном экране. Рассмотрим конкретные функциональные возможности шаблонов.

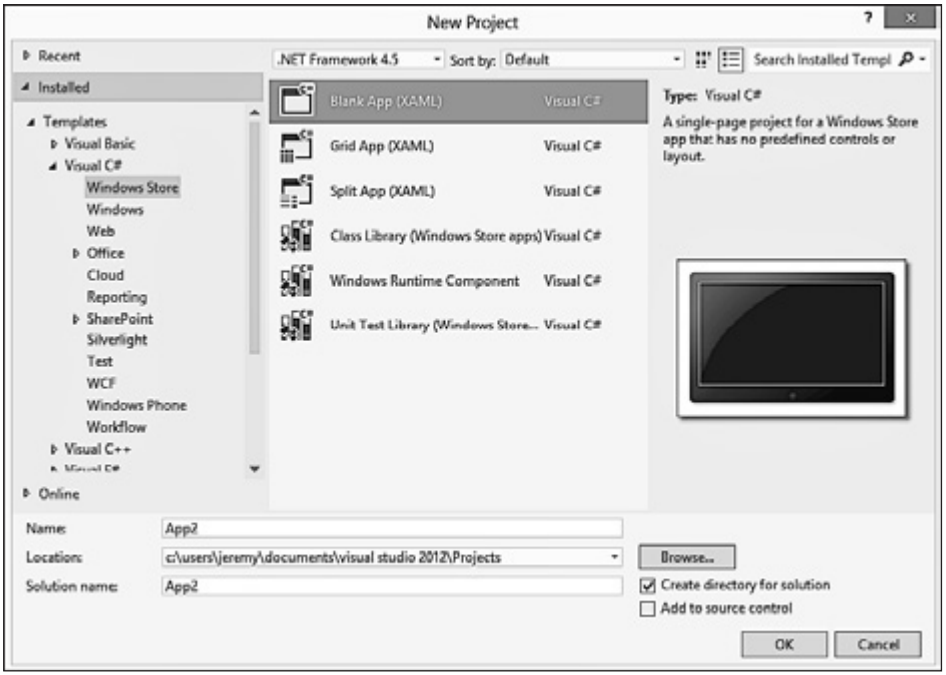


Рис. 2.4. Шаблоны проектов приложений для Магазина Windows на базе Visual C#

Пустое приложение

Применение шаблона Blank App приводит к созданию одностраничного проекта. Здесь нет предопределенных элементов управления или макета. Это простейшая отправная точка, предлагающая базовую инфраструктуру, на основе которой вы можете быстро начать разработку приложения.

Табличное приложение

На основе шаблона Grid App создается проект многостраничного приложения. Приложение поддерживает работу с данными, сгруппированными в категории. Например, приложение для чтения лент новостей может использовать отдельную группу для каждой из лент, а каждая группа может содержать коллекцию элементов или отдельных статей на ленте. Шаблон реализует навигацию по группам и средства детального просмотра контента групп и элементов. На рис. 2.5 показана главная страница приложения, созданного по этому шаблону.

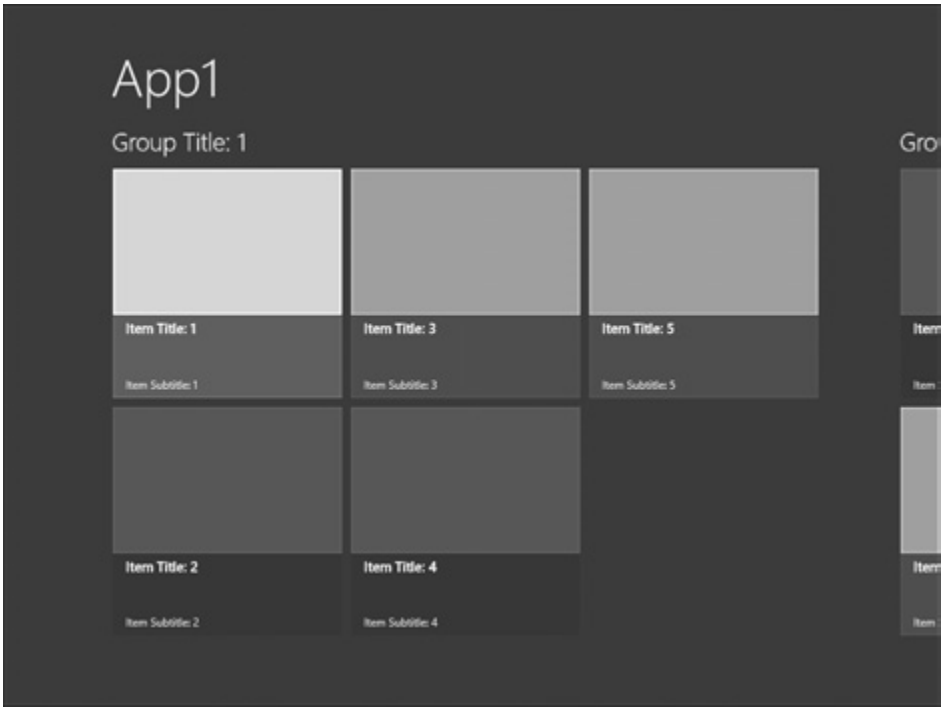


Рис. 2.5. Шаблон табличного приложения

Шаблон проекта предоставляет несколько вспомогательных классов, которые могут вам пригодиться, когда вы будете создавать более сложные приложения.

- `BindableBase`. Базовый класс для моделей представления на основе паттерна MVVM. Он упрощает создание объектов, вовлеченных в систему привязки данных. Дополнительные сведения о привязке данных вы найдете в главе 3.
- `BooleanNegationConverter`. Используется для инвертирования логических значений.
- `BooleanToVisibilityConverter`. Позволяет удобно управлять видимостью элементов пользовательского интерфейса на основе логических значений.
- `LayoutAwarePage`. Базовый класс для объектов типа `Page`, который позволяет прослушивать события смены ориентации и режима просмотра страницы (полноэкранный режим либо состояние фиксации вместе с другим приложением).
- `RichTextColumns`. Задает пользовательский элемент управления, который позволяет связать с ним любое количество элементов данных, автоматически распределяя их по столбцам постоянной ширины.

- `StandardStyles`. Полезный словарь стилей, который помогает настраивать внешний вид приложения. Больше о стилях вы узнаете в главе 3.
- `SuspensionManager`. Вспомогательный класс для сохранения данных. Он особенно полезен для сохранения и восстановления состояния приложения (больше о состоянии приложения вы узнаете в главе 5.

В следующих главах имеется более подробная информация о различных элементах, из которых состоит приложение. Кроме того, вы откроете для себя несколько вариантов совместного использования кода приложениями. Это позволит создавать классы и задействовать их в нескольких приложениях независимо от выбранного шаблона.

Разделенное приложение

Шаблон Split App напоминает Grid App, однако он оптимизирован для просмотра отдельных элементов в списке элементов группы. Начальная страница приложения, созданного по этому шаблону, содержит сводные данные групп. Когда вы приступаете к просмотру контента конкретной группы, экран приложения разделяется. С одной стороны экрана располагается список доступных элементов, а с другой — подробные сведения о выделенном элементе. Разделенный экран приложения, созданного по этому шаблону, показан на рис. 2.6.

Этот шаблон содержит также вспомогательные классы и словари, упомянутые при описании шаблона табличного приложения.

Библиотека классов

Библиотеку классов создают, чтобы ее могли совместно использовать несколько приложений. Шаблон Class Library предлагает минимальный набор конструкций, необходимый для создания сборки, связанной с проектом, построенным по одному из шаблонов приложений. В виде библиотеки классов можно реализовать набор вспомогательных подпрограмм, моделей данных и другой программный код, который нужен вашему приложению.

Компонент среды выполнения Windows

Шаблон Windows Runtime Component служит для создания многократно используемых WinRT-компонентов. Это специализированные библиотеки классов, доступ к которым можно получить из приложений для Windows 8, созданных на базе любых языков. Больше о том, как создавать WinRT-компоненты, вы узнаете в главе 4.

Рис. 2.6. Окно разделенного приложения

Библиотека модульных тестов

Библиотека модульных тестов, создаваемых с помощью шаблона Unit Test Library, содержит специальные классы, описывающие модульные тесты для вашей системы. Методы этих классов вызываются системой тестирования. Тестирование — это важный этап разработки приложений. О тестировании вы узнаете в главе 9.

Приложение ImageHelper

Теперь вы готовы к тому, чтобы создать первое приложение для Windows 8. Разработку этого приложения я начал с шаблона Blank App. Вы можете либо создавать это приложение самостоятельно, следуя за моими пояснениями, либо загрузить его готовую версию с веб-сайта книги http://windows8applications.codeplex.com/. Имя проекта и решения — ImageHelper.

Первый шаг заключается в создании подходящего пользовательского интерфейса. Это можно сделать с применением технологии XAML, о которой рассказывается в главе 3. XAML — это декларативный язык, который

поддерживает создание различных объектов, позволяет задавать их атрибуты и описывать дочерние объекты с помощью расширяемого языка разметки (Extensible Markup Language, XML). Обычно XAML служит для описания UI-элементов, которые определяют внешний вид приложения.

Шаблон по умолчанию генерирует элемент `Grid` (сетка) в файле MainPage.xaml. Следующий шаг заключается в том, чтобы задать две строки в сетке, расположив приведенный код между начальным и конечным тегами элемента `Grid`:

```
<Grid.RowDefinitions>
  <RowDefinition Height="*"/>
  <RowDefinition Height="Auto"/>
</Grid.RowDefinitions>
```

Благодаря этим командам подсистема визуализации данных создаст нижнюю строку, которая автоматически получает размер, необходимый для размещения дочерних элементов. Оставшееся место на экране отдается под верхнюю строку. Это происходит потому, что значение `Auto` указывает на необходимость автоматического подбора размера строки на основе ее контента, а символ звездочки (*) сообщает подсистеме вывода о том, что строка должна занять все оставшееся место.

Далее добавьте команду для вывода изображения-заполнителя после закрывающего тега `</Grid.RowDefinition>`, но перед закрывающим тегом `</Grid>`. Изображение должно быть выровнено по центру верхней строки. Если будет изменен его размер или оно будет растянуто, соотношение сторон должно сохраняться. Изображение выравнивается по центру благодаря соответствующей настройке его свойств, отвечающих за выравнивание; они имеют в названиях слово `Alignment`. Свойство `Stretch` показывает, как именно подсистема вывода должна обрабатывать изменение размера изображения, когда нужно вписать его в имеющееся пространство. Значение этого свойства `Uniform` указывает на то, как следует обрабатывать изображение с учетом соотношения его сторон:

```
<Image x:Name="ImageTarget" Grid.Row="0"
  HorizontalAlignment="Center" VerticalAlignment="Center"
  Stretch="Uniform"/>
```

И наконец, добавьте в разметку элемент `StackPanel`, предназначенный для вывода двух кнопок сразу после только что добавленного элемента `Image`. Этот элемент располагает друг за другом содержащиеся в нем кнопки. В данном случае первая кнопка будет использоваться для захвата нового изображения, вторая — для его сохранения. Благодаря атрибуту `margin` кнопки расположатся на некотором расстоянии друг от друга:

```
<StackPanel Grid.Row="1" Margin="10"
  HorizontalAlignment="Center" Orientation="Horizontal">
  <Button x:Name="CaptureButton" Content="Capture New Image"
    Click="CaptureButton_Click_1"/>
  <Button x:Name="SaveButton" Content="Save Image"
    Click="SaveButton_Click_1" Margin="20 0 0 0"/>
</StackPanel>
```

Если вы сами создаете это приложение, не пользуясь готовым примером, то заметите выпадающее меню, которое появится после того, как вы зададите атрибут `Click` (это происходит благодаря технологии IntelliSense). Выберите в этом меню вариант <New Event Handler>, чтобы код обработчика событий был создан автоматически. Если вы не набираете код, а копируете и вставляете его, удалите значение атрибута `Click` и введите его сами, чтобы увидеть это меню.

Прежде чем вы сможете протестировать созданный интерфейс, вам нужно настроить некоторые параметры приложения. В частности, с их помощью вы можете дать приложению осмысленное имя, которое появится на **начальном** экране, а также указать некоторые режимы и возможности приложения, чтобы завершить работу над ним. Выполните на панели Solution Explorer двойной щелчок на файле Package.appxmanifest. Вы увидите диалоговое окно редактора манифеста приложения, похожее на показанное на рис. 2.7.

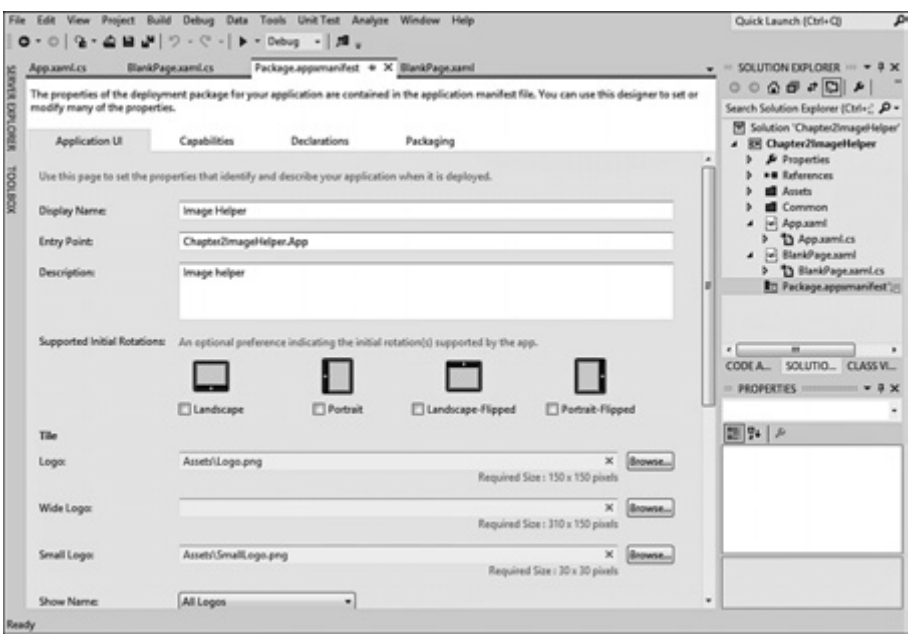


Рис. 2.7. Редактор манифеста приложения

Манифест приложения играет очень важную роль и включает в себя четыре раздела, позволяя менять параметры в этих разделах и сохранять полученные результаты в формате XML. Вы можете просмотреть исходный XML-код манифеста, щелкнув на нем правой кнопкой мыши и выбрав в появившемся меню команду View Code. Манифест содержит следующие разделы:

- Application UI. Параметры, которые влияют на внешний вид приложения. Здесь можно задать осмысленное имя программного продукта, наподобие «Image Helper», и ввести его описание. Вы также можете заблокировать смену ориентации экрана, настроить параметры плиток и уведомлений (больше об этом вы узнаете в главе 7).
- Capabilities. Платформа Windows 8 по умолчанию ограничивает доступные приложениям функциональные возможности и устройства. Если вам нужно работать с подобными ресурсами, вы должны декларировать соответствующую возможность, иначе пользователь увидит сообщение об ошибке при попытке доступа к ограниченному ресурсу. Но даже если вы декларировали возможность, чтобы приложение могло ее задействовать, пользователь должен в явном виде разрешить это, ответив на соответствующий вопрос. Для данного приложения нужно декларировать следующие возможности: Internet (Client), Pictures Library Access и Webcam. Если вы выделите возможность, редактор выведет ее описание, из которого можно больше узнать о ней.
- Declarations. Приложения для Windows 8 могут взаимодействовать друг с другом, опираясь на новые мощные средства, такие как чудо-кнопки и контракты (см. главу 8). При настройке приложения выберите на данной вкладке редактора объявление Share Target и щелкните на кнопке Add. Далее щелкните на кнопке Add New в группе параметров Data Formats и в качестве формата данных введите вариант `Bitmap`. После выполнения этих действий окно редактора должно выглядеть так, как показано на рис. 2.8.
- Packaging. Здесь можно отредактировать уникальные для каждого приложения атрибуты, которые описывают пакет на этапе развертывания. Вы можете задать уникальное имя пакета, отображаемое имя, которое видит пользователь при просмотре состава приложений, установленных в системе, сведения об издателе приложения. Я оставил на данной вкладке имя пакета, предложенное по умолчанию, а в качестве отображаемого задал имя «Image Helper».

Не забудьте сохранить манифест после внесения в него изменений.

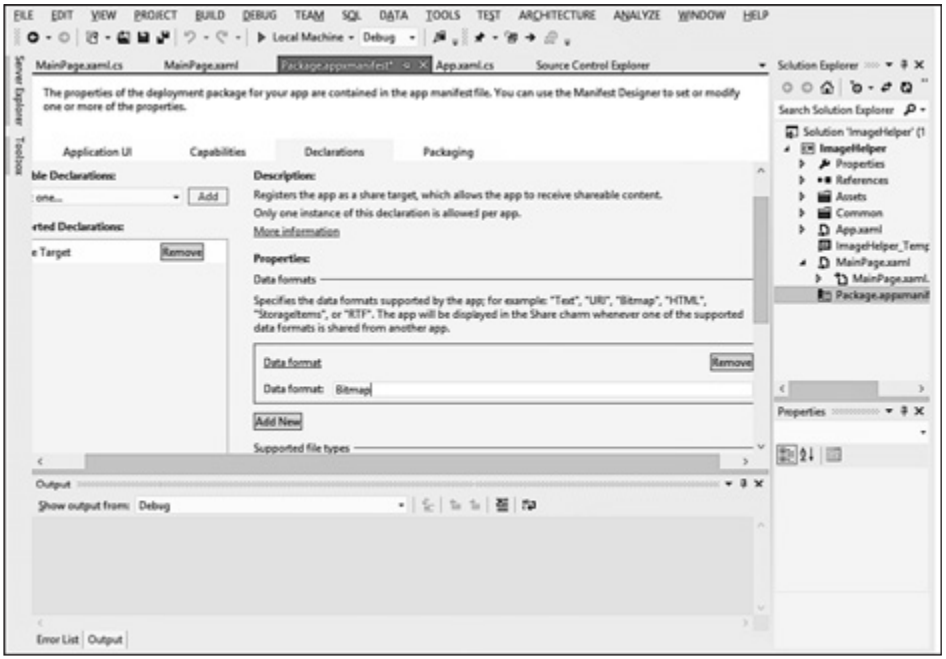


Рис. 2.8. Объявление приложения в качестве конечного и способного принимать растровые изображения

Теперь вы можете написать программный код приложения. Не беспокойтесь, если сейчас что-то покажется вам непонятным. Здесь вы начнете пользоваться некоторыми системными возможностями, а подробности узнаете в следующих главах. Так как приложение было объявлено конечным (Share Target), нам нужно реализовать обработку событий, которые позволят приложению верно реагировать на передачу ему данных из другой программы. В файле App.xaml.cs (это файл вспомогательного кода) добавьте в класс App перегруженный вариант метода `OnShareTargetActivated` со следующим кодом (обратите внимание, что он аналогичен коду запуска приложения, созданному шаблоном; разница только в параметрах, которые он передает странице):

```
protected override void
  OnShareTargetActivated(ShareTargetActivatedEventArgs args)
{
  var rootFrame = new Frame();
  rootFrame.Navigate(typeof(MainPage), args);
  Window.Current.Content = rootFrame;
  Window.Current.Activate();
}
```

Когда пользователь активирует чудо-кнопку Share, выводится список приложений, объявивших себя конечными (Share Target). Для того чтобы увидеть это в действии, откройте приложение, способное отправлять другим приложениям изображения, например Photos, и, удерживая клавишу Windows на клавиатуре, нажмите клавишу C (или проведите пальцем от правого края экрана к центру). На рис. 2.9 вы можете видеть изображение обложки этой книги, выбранное мною для отправки, и список приложений, способных его принять, среди которых есть ImageHelper.

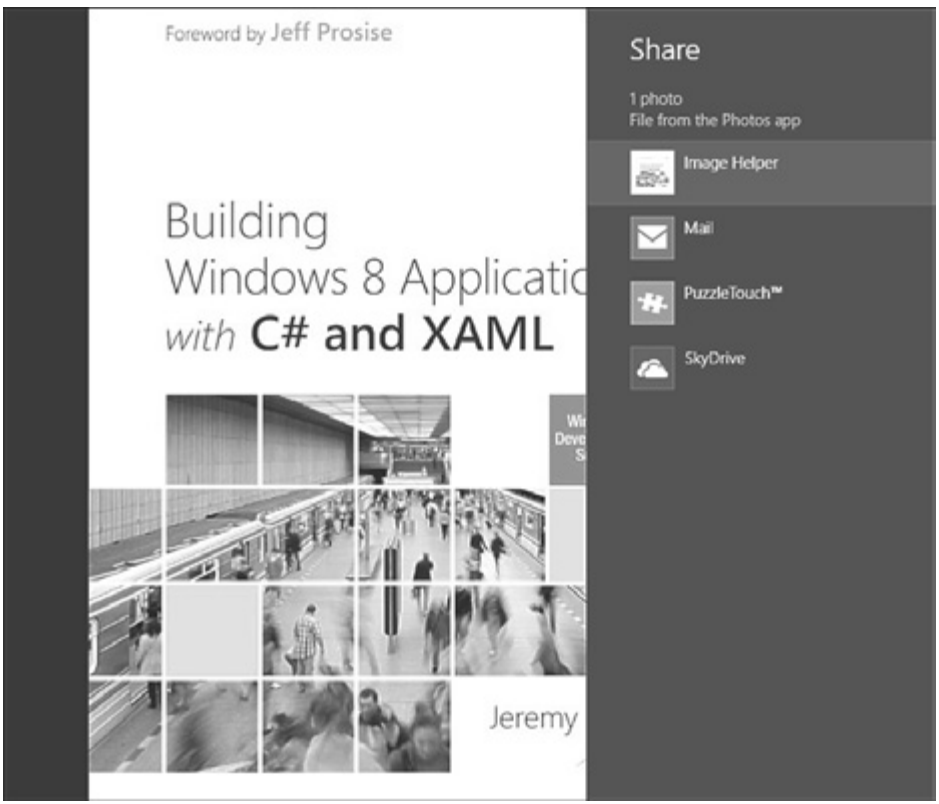


Рис. 2.9. Использование чудо-кнопки Share в приложении Photos

Если пользователь выберет ваше приложение, будет вызвано событие `OnShareTargetActivated`, в параметрах которого имеется информация об объекте, содержащем данные для передачи приложению. Этот код следует за кодом запуска, он создает фрейм для навигации, добавляет в него страницу (это ваша начальная страница), а затем активирует окно и переходит к первой странице.

Вам нужно кое-что сделать и в файле MainPage.xaml.cs. Для начала добавьте закрытое (`private`) поле для хранения объекта `WriteableBitmap` (в верхней

части файла вам понадобится добавить инструкцию `using` для `Windows.UI.Xaml.Media.Imaging`). Данное поле потребуется для хранения изображения, захваченного с камеры, или изображения, отправленного приложению:

```
private WriteableBitmap _writeableBitmap;
```

Далее добавьте код для захвата изображения с камеры. Возможно, вы удивитесь тому, насколько это просто. Добавьте еще две инструкции `using`:

```
using Windows.Media.Capture;
using Windows.Storage.Streams;
```

Код захвата изображения с подключенной к компьютеру веб-камеры приведен в листинге 2.1. Он заменяет пустой метод, автоматически сгенерированный для события щелчка на кнопке (`click`), которое объявлено в XAML-файле. Введите код, если вы создаете приложение «с нуля», затем скомпилируйте, разверните и запустите приложение, нажав клавишу F5. При первом запуске программы вы увидите запрос на разрешение задействовать камеру, как показано на рис. 2.10.

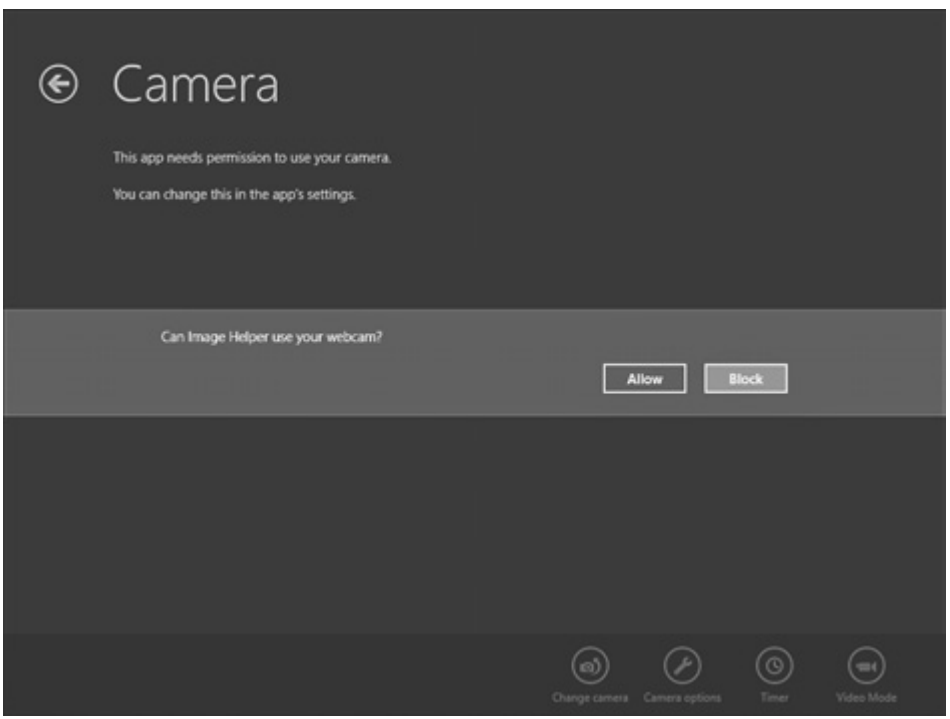


Рис. 2.10. Получение разрешения на использование веб-камеры

Лишь несколькими строками кода вы можете активировать веб-камеру, позволить пользователю сделать снимок и перенести этот снимок в приложение. Если к компьютеру пользователя не подключена веб-камера, появится страница, сообщающая о необходимости подключить веб-камеру. Если же камера подключена, то для того чтобы сделать фото, достаточно просто прикоснуться к изображению, которое вы видите.

Листинг 2.1. Программный код захвата изображения с подключенной к компьютеру камеры

```
private async Task LoadBitmap(IRandomAccessStream stream)
{
  _writeableBitmap = new WriteableBitmap(1, 1);
  _writeableBitmap.SetSource(stream);
  _writeableBitmap.Invalidate();
  await Dispatcher.RunAsync(
  Windows.UI.Core.CoreDispatcherPriority.Normal,
    () => ImageTarget.Source = _writeableBitmap);
}

public async void CaptureButton_Click_1(object sender,
  RoutedEventArgs e)
{
  var camera = new CameraCaptureUI();
  var result = await camera.CaptureFileAsync(
  CameraCaptureUIMode.Photo);
  if (result != null)
  {
    await LoadBitmap(await result.OpenAsync(
      Windows.Storage.FileAccessMode.Read));
  }
}
```

Чтобы закрыть приложение, на сенсорном экране можете либо провести пальцем от верхнего края приложения к нижнему, либо нажать комбинацию клавиш Alt+F4. Это завершит работу приложения, но не остановит отладчик. Для того чтобы это сделать, щелкните на значке Stop в Visual Studio или нажмите клавиши Ctrl+F5.

Приведенный программный код начинается с обращения к системному объекту, который служит для управления камерой. Ключевые слова `async` и `await` используются совместно, чтобы асинхронно вызвать операцию съемки фотографии (вы будете гораздо лучше понимать особенности асинхронного программирования после того, как ознакомитесь с главой 6).

Управление возвратится в наш код, когда пользователь сделает снимок или отменит операцию. Когда пользователь делает снимок, оставшийся код читает данные этого снимка и сохраняет их в специальном объекте типа `WriteableBitmap`. В итоге позже мы можем получить доступ к пиксельным данным изображения (когда пользователь сохранит изображение на диске) для вывода его на экран. Обработка изображения производится в процедуре `LoadBitmap`, ее можно использовать для работы с различными источниками данных.

СОВЕТ

Большинство действий, которые выполняются в Windows 8 с помощью жестов, можно выполнить с помощью мыши и клавиатуры. Например, проведя пальцем от правого края экрана к центру, вы увидите панель чудо-кнопок (Charm Bar). Чтобы достичь того же результата с помощью мыши, переместите указатель в верхний правый или в нижний правый угол экрана. При использовании клавиатуры, удерживая клавишу Windows, нажмите клавишу C. Если вы выполняете отладку приложения и хотите вернуться на рабочий стол, удерживая клавишу Windows, нажмите клавишу D. Если к вашему компьютеру подключено несколько мониторов, и вы хотите указать, на каком из них выводить начальный экран, удерживая клавишу Windows, нажмите клавишу PageUp или PageDown.

Код, который выполняет сохранение изображения, показан в листинге 2.2. Он заменяет обработчик события щелчка на другой кнопке, объявленной в XAML-коде. Данная операция требует следующих директив `using`:

```
using Windows.Storage;
using Windows.Storage.Pickers;
using Windows.Graphics.Imaging;
using System.Runtime.InteropServices.WindowsRuntime;
```

Этот код немного сложнее, но он все равно гораздо проще того, который вы, возможно, применяли раньше. Система задает пользователю вопрос о месте сохранения файла и даже указывает его формат. Все это выполняется посредством WinRT-компонента Picker. Когда пользователь указывает место для сохранения файла, изображение кодируется в формат *PNG* (*Portable Network Graphics*) и сохраняется в указанном месте. Вы можете запустить приложение, сохранить изображение и просмотреть его в проводнике. Больше о работе с изображениями в Windows 8 вы можете узнать на странице http://msdn.microsoft.com/ru-ru/library/windows/apps/br226400.aspx.

Листинг 2.2. Программный код сохранения изображения

```
public async void SaveButton_Click_1(object sender,
  RoutedEventArgs e)
{
  if (_writeableBitmap != null)
  {
    var picker = new FileSavePicker();
    picker.SuggestedStartLocation =
      PickerLocationId.PicturesLibrary;
    picker.FileTypeChoices.Add("Image", new List<string>()
      { ".png" });
    picker.DefaultFileExtension = ".png";
    picker.SuggestedFileName = "photo";
    var savedFile = await picker.PickSaveFileAsync();

    try
    {
      if (savedFile != null)
      {
        IRandomAccessStream output = await
          savedFile.OpenAsync(FileAccessMode.ReadWrite);
        BitmapEncoder encoder =
          await BitmapEncoder.CreateAsync(
            BitmapEncoder.PngEncoderId, output);
          encoder.SetPixelData(BitmapPixelFormat.Rgba8,
          BitmapAlphaMode.Straight,
          (uint)_writeableBitmap.PixelWidth,
          (uint)_writeableBitmap.PixelHeight,
          96.0, 96.0,
          _writeableBitmap.PixelBuffer.ToArray());
          await encoder.FlushAsync();
          await output.GetOutputStreamAt(0).FlushAsync();
      }
    }
    catch (Exception ex)
    {
      var s = ex.ToString();
    }
  }
}
```

Обработчик ошибок в этом коде присутствует лишь для того, чтобы вы могли установить в нем точку останова, если при выполнении программы возникнут какие-либо проблемы. Я постарался сделать наш первый пример

как можно проще, чтобы вы, разбирая его, могли сконцентрироваться на основных особенностях. В нескольких следующих разделах вы узнаете, как Windows 8 хранит информацию о приложении и позволяет осуществлять взаимодействие между приложением и платформой. А сейчас нам осталось выполнить еще один шаг.

Создавая приложение, вы декларировали возможность обмена данными и написали код для вывода принятых от другого приложения данных на главной странице. Получение данных реализует код, приведенный в листинге 2.3. Этот код просто берет данные изображения, переданные нашему приложению, загружает их в переменную типа `WriteableBitmap` и выводит на экран тем же способом, который мы использовали для вывода фотографии, снятой камерой. Данный код находится в файле MainPage.xaml.cs и заменяет пустой метод `OnNavigatedTo`, который по умолчанию генерирует шаблон. Вам нужно будет добавить инструкции `using` для `Windows.ApplicationModel.Activation` и `Windows.ApplicationModel.DataTransfer`, а также объявить поле `_shareOperation`.

Листинг 2.3. Код получения изображения, переданного другим приложением

```
protected async override void OnNavigatedTo(NavigationEventArgs e)
{
  var args = e.Parameter as ShareTargetActivatedEventArgs;
  if (args != null)
  {
    _shareOperation = args.ShareOperation;

    if (_shareOperation.Data.Contains(
      StandardDataFormats.Bitmap))
    {
      _bitmap = await _shareOperation.Data.GetBitmapAsync();
      await ProcessBitmap();
    }
    else if (_shareOperation.Data.Contains(
      StandardDataFormats.StorageItems))
    {
      _items = await _shareOperation.Data
      .GetStorageItemsAsync();
      await ProcessStorageItems();
    }
    else _shareOperation.ReportError(
      "Image Helper was unable to find a valid bitmap.");
  }
}
```

Обратите внимание, что приложение поддерживает графические данные двух форматов. Первый — это необработанные данные изображения. Когда приложение получает изображение, оно сохраняет ссылку на поток с графическими данными, а затем изображение загружается следующим образом:

```
private async Task ProcessBitmap()
{
  if (_bitmap != null)
  {
    await LoadBitmap(await _bitmap.OpenReadAsync());
  }
}
```

Некоторые приложения могут ссылаться на фотоснимки, сохраненные в локальной файловой системе. Вместо того чтобы пакетировать содержимое фотографии в поток для отправки целевому приложению, реализующему операцию общего доступа, приложение создает список элементов хранения данных. Элементы хранения ссылаются на файлы, хранящиеся на устройстве, некоторые из них могут быть файлами изображений. Код в листинге 2.4 перебирает список элементов хранения и загружает первый из них, являющийся графическим файлом.

Листинг 2.4. Перебор элементов хранения данных

```
private async Task ProcessStorageItems()
{
  foreach (var item in _items)
  {
    if (item.IsOfType(StorageItemTypes.File))
    {
      var file = item as StorageFile;
      if (file.ContentType.StartsWith(
        "image",
        StringComparison.CurrentCultureIgnoreCase))
      {
        await LoadBitmap(await file.OpenReadAsync());
        break;
      }
    }
  }
}
```

Скомпилируйте и запустите приложение тем же способом, к которому вы прибегали ранее (нажав клавиши Ctrl+F5). Приложение должно вести себя так же, как в прошлый раз. Теперь выйдите из приложения (нажав клавиши

Alt+F4 или проведя пальцем по сенсорному экрану от верхней части экрана к нижней).

Перейдите к приложению, способному работать с изображениями, — возможно, проще всего воспользоваться системным приложением Photos — и вызовите панель чудо-кнопок. Это можно сделать двумя способами. Если ваше устройство оснащено сенсорным дисплеем, просто проведите большим пальцем от правого края экрана к центру. Если вы пользуетесь мышью, просто переместите указатель в нижний правый угол экрана. В итоге, щелкнув на чудо-кнопке Share, вы увидите список приложений, способных получать изображения от других приложений.

Выберите в появившемся списке приложение ImageHelper. Окно приложения выдвинется с краю экрана. То, что вы увидите, называется *всплывающим элементом* (fly-out), так как окно перекрывает видимое окно другого приложения. Наше приложение использует свою главную страницу, чтобы упростить вывод изображения. Однако в большинстве случаев вы будете разрабатывать для реализации подобного механизма специальную страницу, которая занимает меньше места на экране. То, что отобразится на появившейся странице, представляет собой эскиз переданного изображения с теми же кнопками, которые используются для захвата изображения с камеры или для сохранения изображения на диске. Это та же самая страница, которая применяется для захвата изображения с камеры.

Данный пример упрощен, его цель — демонстрация операции получения изображения от другого приложения. Когда вы щелкнете на одной из кнопок, окно приложения автоматически скроется. Вы не сможете сохранить файл, так как окно выбора файлов пытается расположиться поверх окна приложения. Это запрещено для окон, представленных всплывающими элементами. О том, как обрабатывать операции общего доступа к данным и как соответствующим образом реализовывать подобные сценарии, вы узнаете в главе 8.

Как это работает

Наше первое приложение обладает неплохой функциональностью! Вы с его помощью можете не только сделать снимок, воспользовавшись вебкамерой, подключенной к компьютеру, но и сохранить его. Вы также можете получить изображение от другого приложения и просмотреть его в своей программе. Если вы вошли в систему с учетной записью сервиса SkyDrive, то сможете сохранять изображения в облачном хранилище данных (оно будет присутствовать в качестве одного из вариантов в диалоговом окне выбора места сохранения файла) без необходимости изменения кода приложения. Как же Windows 8 обеспечивает подобное взаимодействие между приложениями? Ответ заключается в особенностях WinRT-компонентов.

Приложения — это WinRT-компоненты

Вы узнали об архитектуре WinRT в предыдущей главе. Вы также использовали некоторые WinRT-компоненты в вашем первом приложении. `CameraCaptureUI` — один из примеров подобного компонента, как и `FileSavePicker`. Вы, вероятно, удивитесь, когда узнаете, что приложение ImageHelper, которое вы создали, тоже является WinRT-компонентом! Приложения — это особые WinRT-компоненты, которые могут быть запущены либо с начального экрана, либо при обслуживании соответствующего контракта.

Жизненный цикл приложения для Windows 8 может быть сведен к трем простым фазам: установка, запуск, исполнение. В фазе установки выполняется развертывание компонентов приложения в операционной системе. Приложение регистрируется в системе (в том числе регистрирует объявления и возможности). Когда пользователь прикасается к плитке приложения на начальном экране, Windows 8 находит компонент, соответствующий приложению, и запускает его. То есть запускает приложение.

Расширения и классы

Возможно, вы удивитесь, узнав о том, как именно Windows 8 хранит информацию о ваших приложениях. Операционная система использует реестр Windows. Реестр — это иерархическая база данных, в которой Windows хранит различные параметры и сведения о предпочтениях пользователя. Реестр впервые появился в Windows 3.1, то есть ему уже больше 20 лет! Эта база данных отвечает за особенности взаимодействия пользователя с Windows 8, в том числе за контракты, с помощью которых приложения взаимодействуют друг с другом.

Когда вы устанавливаете приложения, точка входа в программу сохраняется в реестре в качестве класса. Классы — это уникальные идентификаторы WinRT-компонентов, в том числе приложений. Расширения, в свою очередь, описывают контракты. В системе существует множество контрактов, и все они содержат коллекции классов, которые могут взаимодействовать с контрактом.

Для того чтобы увидеть, как все это работает, откройте командную строку. Проще всего это сделать, нажав клавишу Windows, чтобы попасть на начальный экран, затем введя с клавиатуры команду `cmd` и щелкнув мышью на появившемся значке Command Prompt. Введите в командной строке команду `regedit` и нажмите клавишу Enter. На запрос о разрешении выполнения данной операции, поступивший от системы контроля учетных записей пользователей, ответьте Yes. Будет открыт классический редактор реестра, который существовал и в других версиях Windows. Также его можно запустить, используя механизм поиска, то есть нажав клавишу Windows, введя команду regedit и щелкнув на появившемся значке.

Разверните в редакторе реестра ветвь HKEY_CURRENT_USER ▸ Software ▸ Classes ▸ ActivatableClasses ▸ Package. Вы увидите запись, которая начинается с имени пакета, — это та самая длинная строка, которая выводится в поле Package name на вкладке Packing редактора манифеста. Данная запись в реестре соответствует тому приложению, которое вы только что создали и развернули в системе. Вы можете видеть сведения об имени пакета и версии, региональные параметры и сгенерированный код, соответствующий приложению. Так Windows 8 получает сведения о том, как следует запускать ваше приложение (например, приложения на базе HTML должны запускаться с использованием прикладного хост-процесса, предоставляющего функциональность веб-браузера) и какие класс и метод вызвать, чтобы это сделать.

Теперь разверните в реестре ветвь HKEY_CURRENT_USER ▸ Software ▸ Classes ▸ Extensions ▸ ContractId. Здесь описываются системные контракты. Вы можете узнать там некоторые из них, такие как Windows.ShareTarget и Windows.File. Есть здесь и еще один контракт, который называется Windows.Launch. Это контракт для WinRT-компонентов, которые можно запустить с начального экрана. Если вы развернете данный раздел реестра, то увидите там несколько элементов, среди которых должно присутствовать и приложение ImageHelper. Оно же есть в разделе реестра Windows.ShareTarger.

Созданное вами приложение — это особый WinRT-компонент, сведения о котором Windows 8 хранит в реестре. Когда пользователь активирует чудо-кнопку Share, Windows 8 открывает реестр, находит там список приложений, которые могут выступать в качестве целевых для операции обмена данными, и показывает этот список пользователю. Когда пользователь выбирает одно из приложений, класс, ссылка на который есть в реестре, позволяет Windows определить, какой компонент нужно создать и какой метод вызвать. То же происходит, когда приложение запускает пользователь. Плитка приложения — это нечто вроде особой чудо-кнопки запуска, при обслуживании которой используется тот же подход с поиском нужного класса и запуском компонента.

Дизассемблер промежуточного языка

В дополнение к регистрации информации о приложении для системных целей, программы для Windows 8 публикуют метаданные. Эти метаданные описывают интерфейсы, классы, методы и свойства, предоставляемые WinRT-компонентами и приложениями. Для приложений на базе HTML5, C++ или C# публикуются метаданные одного и того же типа. Это упрощает исследование ваших компонентов другими программами, которые могут узнать, как пользоваться их компонентами. Это также позволяет проецировать возможности компонентов в другие языки. В итоге из других языков можно пользоваться компонентами так, как если бы они были написаны на этих языках.

Вы можете задействовать приложение IDLASM, чтобы исследовать метаданные компонентов. Для того чтобы увидеть данный инструмент в действии, откройте командную строку разработчика, открыв панель чудо-кнопок, как показано на рис. 2.11.

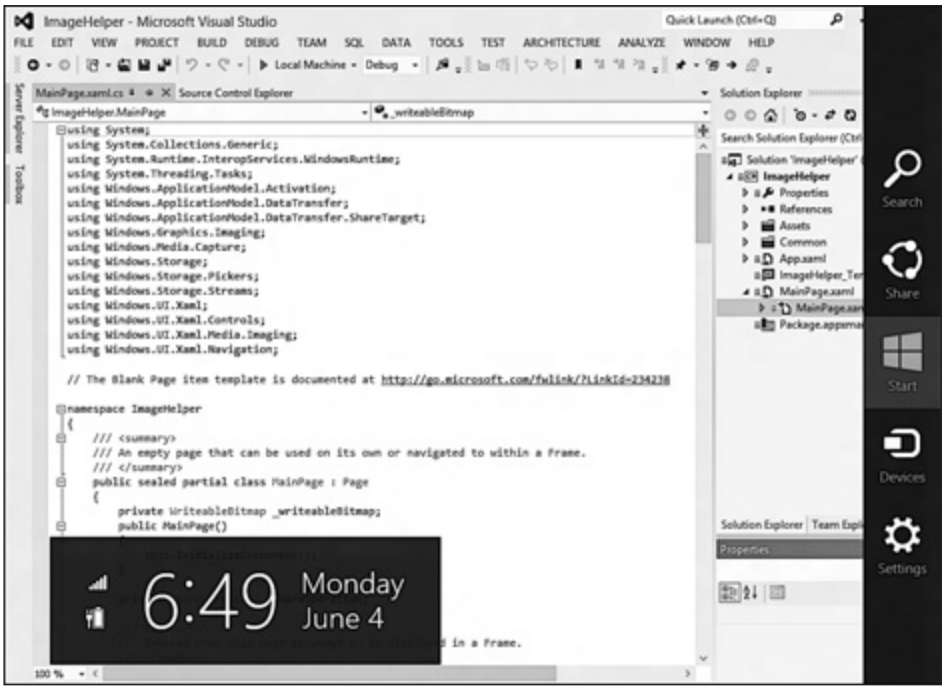


Рис. 2.11. Вызов панели чудо-кнопок в ходе работы

Когда панель появится, активируйте чудо-кнопку Search и введите слово `Command`. В результате должен появиться список найденных объектов, включая объект Developer Command Prompt. Для того чтобы упростить использование командной строки разработчика, щелкните на ее значке правой кнопкой мыши для вывода панели приложения. Затем щелкните мышью на значке Pin to taskbar, введите в командной строке текст `ildasm.exe` и нажмите клавишу Enter.

Эта приведет к запуску дизассемблера промежуточного языка .NET Framework (.NET Framework IL Disassembler). Его обычно используют для просмотра .NET-сборок и соответствующего им кода на промежуточном языке (Intermediate Language, IL). В случае Windows 8 он может применяться для исследования WinRT-приложений и WinRT-компонентов, поскольку приложения для Windows 8 соответствуют стандарту ECMA-335. Это открытый стандарт, опубликованный Microsoft. Узнать подробности об этом стандарте вы можете на странице http://www.ecma-international.org/publications/standards/Ecma-335.htm.

В Visual Studio 2012 щелкните правой кнопкой мыши на имени проекта ImageHelper в окне Solution Explorer, чтобы открыть контекстное меню. Выберите в нем пункт Open Folder in Windows Explorer. В результате папка, в которой расположено приложение, откроется в проводнике. Раскройте подпапку bin ▸ Debug, чтобы увидеть файлы, полученные после компиляции приложения. Содержимое папки должно быть похожим на рис. 2.12.

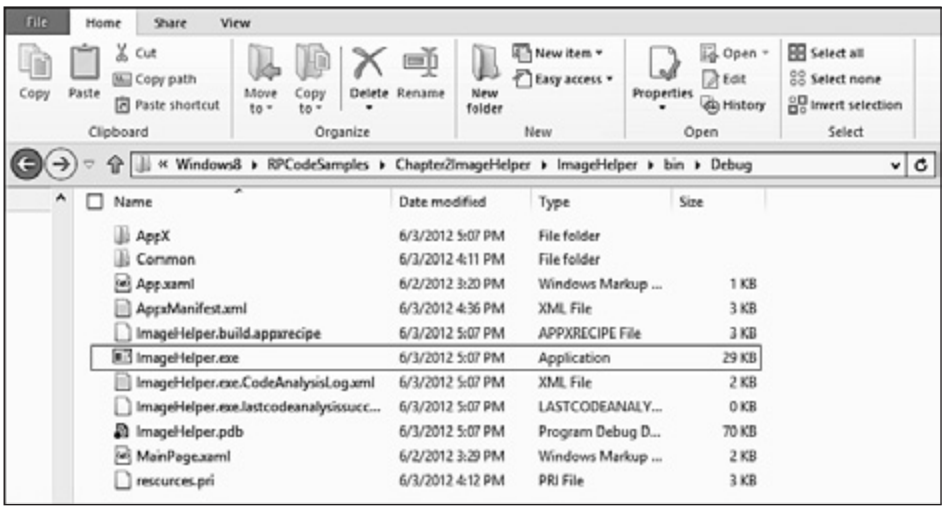


Рис. 2.12. Файлы приложения для Windows 8

Выделите щелчком файл ImageHelper (на рисунке видны расширения файлов — так настроена моя система, в вашем случае они могут быть скрыты, поэтому при выделении файла ориентируйтесь на его тип, который должен иметь значение Application) и перетащите его в окно приложения ILDASM. Вы увидите дерево элементов, которое является результатом синтаксического разбора метаданных исполняемого файла. Вы можете разворачивать узлы этого дерева и просматривать информацию о классах, методах, свойствах и событиях, которые созданы для приложения. Это показано на рис. 2.13. Как видите, приложение, написанное на C#, на самом деле генерирует IL-код и использует .NET Framework. WinRT-компоненты операционной системы проецируются в приложение и выглядят как собственные классы приложения.

Наше приложение использует несколько основных WinRT-компонентов. Даже если компоненты являются частью операционной системы, с помощью дизассемблера можно просматривать их сигнатуры. Например, чтобы осуществить захват изображения с веб-камеры, мы включили в код ссылку на компонент Windows.Media.Capture и использовали компонент CameraCaptureUI. Вы можете исследовать эти компоненты, если знаете, где именно они расположены.

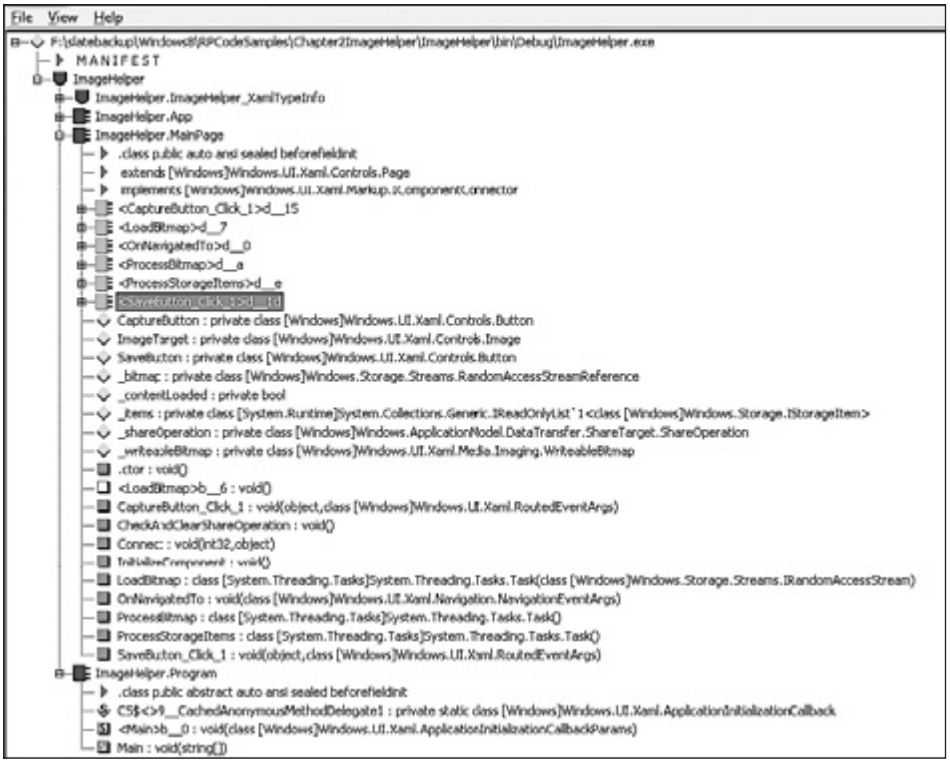


Рис. 2.13. Внутренние метаданные и код приложения ImageHelper

В окне программы ILDASM либо выполните команду File ▸ Open, либо нажмите клавиши Ctrl+O. В появившемся диалоговом окне открытия файлов перейдите к папке метаданных Windows 8: C:\windows\system32\WinMetadata.

Там вы увидите файлы с расширением .wimd. Эти файлы содержат сведения о метаданных WinRT-компонентов, их имена не зависят от языка, на котором написан компонент. Прокрутите список файлов вниз и выберите файл Windows.Media.winmd (расширение будет скрыто, если вы не настроили соответствующим образом свойства папки, но в диалоговом окне должен отображаться тип файла, в данном случае — WINMD File). Сделайте двойной щелчок на файле или, выделив его, щелкните на кнопке Open. На рис. 2.14 вы можете видеть несколько развернутых узлов в структуре компонента. Здесь хорошо видно объявление класса и метод, используемый для захвата изображения.

Вы можете получить представление об устройстве платформы Windows 8, исследовав файлы в папке метаданных. Это лишь встроенные WinRT-компоненты. С помощью вашего программного кода и других приложений, установленных в системе, можно получить доступ к гораздо большему количеству компонентов. Вы можете отчетливо видеть, что платформа Windows 8 построена на технологиях, которые имеют многолетнюю историю

Рис. 2.14. Метаданные WinRT-компонента

и с которыми знакомы многие разработчики. И она объединяет лучше из них, что делает ее столь мощной и в то же время простой в использовании.

Выводы

В этой главе вы изучили несколько вариантов настройки рабочего пространства разработки приложений для Windows 8. Вы узнали о различных инструментах разработки и о шаблонах, которые предоставляет Visual Studio 2012. Кроме того, вы создали учебное приложение, которое способно захватывать фотографии, сделанные веб-камерой, получать изображения от других приложений и сохранять изображения в локальной файловой системе или в облачном хранилище данных. И все это с помощью всего нескольких десятков строк кода.

После того как вы создали и развернули приложение, вы узнали о том, как Windows 8 использует системный реестр для хранения информации о приложениях и о контрактах, которые они реализуют. Также вы узнали об открытых стандартах, в соответствии с которыми приложения для Windows 8 предоставляют метаданные компонентов, и о приложении ILDASM, позволяющем просматривать эти метаданные. В следующей главе вы ближе познакомитесь с технологией XAML и узнаете, как с ее помощью строить пользовательский интерфейс приложений для Windows 8.

Расширяемый язык разметки приложений 3

В предыдущей главе мы разработали приложение с простым пользовательским интерфейсом, который был описан в файле с расширением .xaml. XAML — это аббревиатура словосочетания Extensible Application Markup Language (Расширяемый язык разметки приложений). Этот язык получил распространение, благодаря его использованию в приложениях для платформ Windows Presentation Foundation (WPF) и Silverlight. Разработчики, которые раньше имели дело с XAML, с легкостью смогут применить свои знания при создании приложений для Windows 8.

XAML — это разработанный Microsoft декларативный язык разметки на базе XML. Часто его ошибочно называют языком пользовательского интерфейса и сравнивают с HTML. Хотя XAML играет главную роль в разработке пользовательского интерфейса приложений для Windows 8, его возможности не ограничены созданием элементов интерфейса для платформ наподобие Silverlight или WPF. На самом деле XAML используется и в других технологиях, в том числе в Windows Workflow Foundation (WF).

XAML представляет собой язык представления структурированной информации. Он оперирует тремя ключевыми понятиями: объекты, члены и текст. В основе XAML-объектов лежат либо CLR-типы, экземпляры которых в XAML создаются подсистемой синтаксического разбора, либо вызовы WinRT-компонентов. В основе XAML-членов лежат свойства типов, а текст служит для задания значений. XAML-разметку можно представить в виде дерева объектов, описанных с помощью языка XML. В следующем разделе вы узнаете, что XAML позволяет декларативно создавать экземпляры объектов.

Исходный код примеров к этой книге можно найти на странице http://windows8applications.codeplex.com/.

Декларирование пользовательского интерфейса

При разработке приложений для Windows 8 на языке C#, VB или C++ пользовательский интерфейс описывают с помощью XAML. На самом деле, XAML — это набор объявлений объектов, обрабатываемый приложением для Windows 8. На его основе строится пользовательский интерфейс, который функционирует так, как нужно разработчику. Взгляните на листинг 3.1. Это полная XAML-разметка пользовательского интерфейса приложения ImageHelper.

Листинг 3.1. Декларативное описание пользовательского интерфейса на языке XAML

```
<Page
  x:Class="ImageHelper.MainPage"
  IsTabStop="false"
  xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
  xmlns:local="using:ImageHelper"
  xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
  xmlns:mc="http://schemas.openxmlformats.org/markup-
      ↳ compatibility/2006"
  mc:Ignorable="d">
  <Grid Background="{StaticResource ApplicationPageBackgroundThemeBrush}">
    <Grid.RowDefinitions>
      <RowDefinition Height="*"/>
      <RowDefinition Height="Auto"/>
    </Grid.RowDefinitions>
    <Image x:Name="ImageTarget" Grid.Row="0"
      HorizontalAlignment="Center" VerticalAlignment="Center"
      Stretch="Uniform"/>
    <StackPanel Grid.Row="1" Margin="10"
        HorizontalAlignment="Center" Orientation="Horizontal">
        <Button x:Name="CaptureButton"
      Content="Capture New Image"
      Click="CaptureButton_Click_1"/>
        <Button x:Name="SaveButton" Content="Save Image"
      Click="SaveButton_Click_1" Margin="20 0 0 0"/>
    </StackPanel>
  </Grid>
</Page>
```

XAML-разметка указывает подсистеме синтаксического разбора на то, что нужно создать объект `Page` (страница). Этот объект имеет определенные свойства. Например, в нем используется ссылка на класс `MainPage`, показывающая местоположение программного кода страницы. У страницы есть дочерний элемент `Grid` (сетка), который описывает особенности расположения других объектов на экране. Сетка содержит ссылку на ресурс, содержащий цвет фона. О ресурсах мы поговорим в этой главе далее. Объект `Image` служит для вывода на экран фотографии, снятой веб-камерой, или изображения, полученного из другого приложения. Элемент `Button` (кнопка) не только отвечает за вывод на экран графического представления кнопки, но и содержит ссылку на обработчик события щелчка на кнопке.

Все элементы в примере, приведенном в листинге 3.1, можно создать программно. То есть можно создать экземпляр объекта `Page`, создать новый объект `Grid` и назначить его странице и т. д. Язык XAML упрощает построение макета страницы приложения и предоставляет возможность визуального проектирования интерфейса. В таком режиме можно настраивать интерфейс и тут же видеть, что получилось.

В верхней части XAML-кода содержится несколько важных объявлений, касающихся *пространств имен*. В XAML и XML пространства имен, предоставляя области видимости, обеспечивают уникальность имен элементов и атрибутов. `x:Class` — это специальный атрибут в XAML-пространстве имен (оно задано частью `x:` в имени). Этот атрибут задает объявление частичного класса и, таким образом, может быть отображен на файл вспомогательного кода (code-behind). Пространство имен `xmlns:` — это по умолчанию XML-пространство имен. Вы можете использовать его для объявления собственных областей видимости. Для того чтобы назначить пространство имен `MyApp.Mynamespace` префиксу `local`, достаточно добавить такое объявление:

```
xmlns:local="using:MyApp.MyNamespace"
```

Затем можно обращаться к типу `MyType`, который расположен в пространстве имен `MyApp.Mynamespace`, следующим образом:

```
<local:MyType/>
```

Для того чтобы объявить область видимости некоего пространства имен, нужна ссылка на сборку, которая содержит это целевое пространство имен. Для того чтобы некий тип был успешно получен из XAML-разметки, в проекте должна быть ссылка на сборку, которая содержит этот целевой тип.

Вы могли заметить, что элементы, которые следуют за объявлениями пространств имен в XAML, имеют иерархическую организацию. Существует

родительский объект, который содержит дочерние объекты. Они, в свою очередь, тоже могут включать в себя дочерние объекты. Некоторые элементы могут быть отключены или настроены так, чтобы их не было видно на экране. Элементы, видимые на экране, формируют особую иерархию, которую называют *визуальным деревом*.

Визуальное дерево

Визуальное дерево представляет собой описание видимых UI-элементов. На рис. 3.1 схематично показано визуальное дерево для нашего простого приложения ImageHelper.

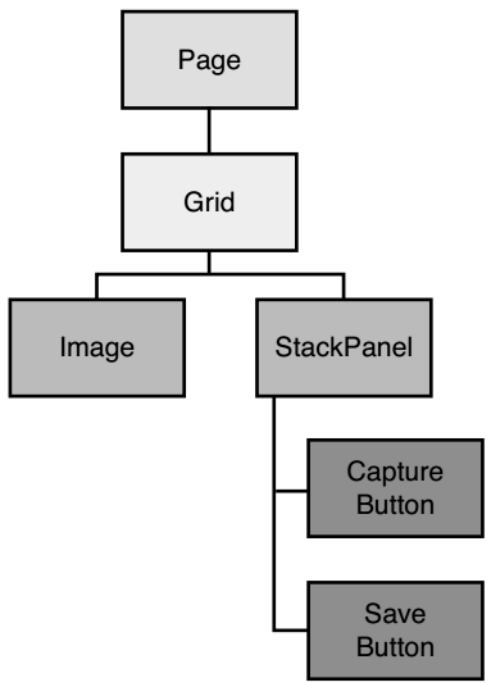


Рис. 3.1. Визуальное дерево

Важно, чтобы вы понимали концепцию визуального дерева, так как она связана с вызовом UI-элементами специальных событий, которые называются *перенаправляемыми* (routed events). Для того чтобы понять, зачем нужны перенаправляемые события, представим себе, что происходит, когда пользователь касается некоторой области экрана, в которой отображается кнопка. Какой UI-элемент должен отреагировать на данное событие?

С технической точки зрения кнопка — это часть элемента `StackPanel`, а сам этот элемент является дочерним для элемента `Grid`. Пользователь на самом деле одновременно касается всех трех элементов.

Когда это происходит, вызывается событие и происходит его *подъем* (bubble up) по визуальному дереву. Событие обрабатывается элементом дерева с наибольшим уровнем вложенности (то есть самым нижним). Для того чтобы увидеть, как это работает, загрузите пример **ImageHelper2** для главы 3 с веб-сайта с примерами для книги. В этом примере к элементам `Grid` и `StackPanel` добавлено событие `PointerPressed`. Кроме того, здесь имеется красный прямоугольник (элемент `Rectangle`), который расположен между двумя кнопками:

```
<Rectangle Width="30" Height="30" Fill="Red"
  Margin="20 0 0 0"
  PointerPressed="Rectangle_PointerPressed_1"/>
```

При добавлении события к XAML-элементу в редакторе разметки можно просто нажать клавишу Tab, чтобы был автоматически сгенерирован *вспомогательный код обработчика этого события* (code-behind event handler). Затем элемент «слушает», не произошло ли чего-нибудь. Когда это событие происходит, вызывается его обработчик. Обработчик, в свою очередь, может выполнить какие-либо действия. Например, вызвать интерфейс веб-камеры, чтобы пользователь мог сделать снимок, или открыть диалоговое окно сохранения файла. Событие `PointerPressed` вызывается, когда пользователь касается экрана пальцем или пером либо щелкает левой кнопкой мыши. Windows 8 упрощает разработку, позволяя единообразно обрабатывать ввод данных с помощью различных устройств.

В нашем случае код, который вызывается для обработки события, выводит текст в окно отладочной информации и в текстовое поле на странице:

```
private void ShowPointerPressed(string source)
{
  var text = string.Format("Pointer pressed from {0}", source);
  Events.Text = string.Format("{0} // {1}", Events.Text, text);
  Debug.WriteLine(text);
}
```

Скомпилируйте и запустите приложение в отладчике (нажав клавишу F5). Щелкните на красном прямоугольнике мышью или прикоснитесь к нему. Что произошло? На экране и в окне отладочной информации вы должны увидеть сведения о трех событиях (если вы не видите этого окна, нажмите клавиши Ctrl+Alt+O).

```
Pointer pressed from Rectangle // Pointer pressed from StackPanel
↪ // Pointer pressed from Grid
```

Именно это и называется подъемом события. Событие перемещается по визуальному дереву, от элемента `Rectangle` через содержащий его элемент `StackPanel` к родительскому элементу `Grid`. Теперь щелкните на пустом пространстве между прямоугольником и одной из соседних кнопок. Результат вас удивил? Можно было бы ожидать, что событие вызовет элемент `StackPanel`, а затем — элемент `Grid`. Однако в окне отладочных данных видно лишь событие элемента `Grid`. Почему?

XAML-движок достаточно интеллектуален, чтобы распознать «пустое пространство» и не вызывать для него событие. Предполагается, что свободное пространство элемента `StackPanel` не должно реагировать на событие касания. У элемента `StackPanel` не задан фоновый цвет, он прозрачен, и любой его свободный участок, которого коснется пользователь, просто «пропустит» событие к родительскому элементу. Назначьте элементу `StackPanel` серый фон, добавив к его объявлению выделенный код:

```
<StackPanel Grid.Row="1" Margin="10"
  Background="Gray"
  PointerPressed="StackPanel_PointerPressed_1"
  HorizontalAlignment="Center" Orientation="Horizontal">
```

Если вам нужно, чтобы «пустое пространство» обрабатывало события, можете задать для фона элемента значение `Transparent` (это прозрачный цвет). Внешне такой элемент не будет отличаться от элемента, фоновый цвет которого не задан, однако в поведении элементов появится одно важное различие. Фон будет представлен прозрачной поверхностью, которая может обрабатывать события. В итоге незаполненное пространство элемента сможет реагировать на касание, удержание и другие жесты.

Снова запустите приложение и прикоснитесь к серой фоновой области, не занятой кнопкой или прямоугольником. Вы увидите, что элемент `StackPanel` теперь получает событие. Теперь щелкните на кнопке Save Image (Сохранить изображение). Что произошло? Вы снова удивлены? (Никаких событий не происходит.) Не удивляйтесь, дальнейший подъем события вверх по дереву можно остановить. На самом деле событие `Click` (щелчок) кнопки — это краткое имя события `PointerPressed` (нажатие), а так как кнопка сама обрабатывает это событие, она предотвращает его дальнейшее распространение по дереву.

Откройте файл вспомогательного кода MainPage.xaml.cs и добавьте выделенный курсивом код в обработчик события `PointerPressed` элемента `StackPanel`:

```
private void StackPanel_PointerPressed_1(object sender,
  Windows.UI.Xaml.Input.PointerRoutedEventArgs e)
{
  ShowPointerPressed("StackPanel");
  e.Handled = true;
}
```

ПРИМЕЧАНИЕ

На различных планшетных компьютерах, которые работают под управлением Windows 8, применяются разные способы взаимодействия с приложениями. Кнопку можно «нажать», щелкнув на ней мышью либо коснувшись ее пальцем или пером. В этой книге всегда, когда вам встречаются слова «щелкнуть», «нажать» или «коснуться», предполагается, что вы получите один и тот же результат, используя левую кнопку мыши, палец или перо для сенсорного ввода данных. Если я ожидаю, что вы примените какой-то конкретный способ, я упоминаю об этом. Там, где это возможно, в качестве эквивалентов соответствующих сенсорных жестов я привожу описания подходящих клавиатурных сокращений и команд, которые можно выполнить с помощью мыши.

Снова скомпилируйте и запустите приложение. Щелкните либо на прямоугольнике, либо на пустом пространстве, которое его окружает. Теперь вы должны увидеть сведения о событии, поступившие от элемента `StackPanel`, но не от элемента `Grid`. Это происходит потому, что элемент `StackPanel` помечает событие как обработанное, чтобы предотвратить его дальнейший подъем по визуальному дереву.

Зависимые свойства

Большинство XAML-объектов являются наследниками базового класса `DependencyObject`. Это класс, который позволяет объектам участвовать в работе *системы зависимых свойств* (dependency property system). Данная система расширяет возможности традиционных свойств; зависимые свойства участвуют в отношениях наследования, могут оказывать влияние на другие классы и управляются особым образом.

Извне зависимое свойство выглядит как статическое поле, зарегистрированное в системе зависимых свойств. В соответствии с соглашением об именовании имя поля должно заканчиваться словом `Property`. Для регистрации свойства вызывают статический метод класса `DependencyProperty`. Для того чтобы увидеть, как это работает, откройте проект Dependency-

Properties. Взгляните на код класса `MyDependencyObject`, потомка класса `DepencencyObject`:

```
public class MyDependencyObject : DependencyObject
```

В данном случае зависимое свойство просто хранит целое число. Оно реализуется с помощью статического объекта типа `DependencyProperty`, например, так:

```
public static readonly DependencyProperty
  MyNumberProperty = DependencyProperty.Register(
    "MyNumber",
    typeof (int),
    typeof (MyDependencyObject),
    new PropertyMetadata(2, OnMyNumberPropertyChange));
```

Имя свойства в соответствии с соглашением об именовании оканчивается словом `Property`. Первый параметр в вызове статического метода `Register` содержит имя свойства (без окончания `Property`). Второй параметр задает тип свойства, в данном случае — это целое (`int`). Третий параметр задает тип владельца свойства — класса, от которого свойство зависит. Последний параметр не обязателен, он может содержать значение свойства, предлагаемое по умолчанию, и (или) метод, который нужно вызывать, когда свойство меняется.

В нашем случае код метода, который вызывается при изменении свойства, выводит новое значение свойства в окне отладочной информации:

```
private static void OnMyNumberPropertyChange(DependencyObject d,
  DependencyPropertyChangedEventArgs e)
{
  Debug.WriteLine("Property changed: {0}", e.NewValue);
}
```

Файл MainPage.xaml содержит ссылку на этот новый зависимый объект. В XAML-коде есть объявление элемента управления `Slider` (ползунок). Он предназначен для задания значения зависимого объекта. Этот объект описан в контексте данных сетки:

```
<Grid.DataContext>
  <local:MyDependencyObject/>
</Grid.DataContext>
```

Ниже описания контекста данных элемента `Grid` находится описание элемента управления `Slider`. Он привязан к свойству `MyNumber` зависимого

объекта. Привязка в данном случае означает, что начальное значение ползунок получает из связанного с ним объекта. Когда ползунок перемещается, система привязки данных обновляет целевое свойство, так как в данном случае используется двусторонняя привязка:

```
<Slider Minimum="1" Maximum="100" Height="20"
  HorizontalAlignment="Stretch"
  Value="{Binding Path=MyNumber,Mode=TwoWay}"
  x:Name="Slider"/>
```

Подробнее о привязке данных мы поговорим далее в этой главе.

ПРИМЕЧАНИЕ

Особое зависимое свойство DataContext (контекст данных) мы подробно рассмотрим в разделе «Привязка данных» этой главы. Данное свойство может быть представлено любым объектом, но чаще всего это класс, являющийся наследником класса DependencyObject или реализующий интерфейс INotifyPropertyChanged. И то и другое позволяет объекту отправлять уведомления об изменениях свойства, которые использует система привязки данных. Свойство DataContext уникально не только потому, что задает базовый объект привязки данных, но и потому, что оно наследуется. Когда вы задаете свойство DataContext на уровне корневого элемента Grid, все дочерние элементы используют тот же самый контекст данных, если только он в них не переопределяется.

Хотя для регистрации зависимых свойств служит система зависимых свойств, среде исполнения они должны представляться в виде обычных CLR-свойств. Это называют «резервированием» свойств. Значение может быть представлено в виде обычного свойства, но с системой зависимых свойств будут работать методы чтения (get) и установки (set) значения свойства. Вот код «резервирования»:

```
public int MyNumber
{
  get { return (int) GetValue(MyNumberProperty); }
  set { SetValue(MyNumberProperty, value);}
}
```

Методы `GetValue` и `SetValue` предоставлены базовым классом `DependencyObject`. Они применяются для запроса текущего значения свойства или для его установки из «резервного» CLR-свойства. Запустите программу в режиме отладки и переместите ползунок. Вы должны увидеть, как в окне вывода отладочных данных появятся новые значения, что является результатом

заданного вами обратного вызова, выполняемого при изменении значения свойства.

Зависимые свойства имеют некоторые преимущества по сравнению с обычными CLR-свойствами. Вот некоторые из этих преимуществ.

- **Уведомление об изменении свойства.** Зависимые свойства автоматически сообщают системе привязки данных, когда их значения изменяются.
- **Реализация обратного вызова при изменении свойства.** Разработчик может задать делегат, который будет принимать обратные вызовы при изменении свойств, что позволяет распространять изменения за пределы системы привязки данных.
- **Значения, предлагаемые по умолчанию.** Зависимые свойства могут определяться со значением, предлагаемым по умолчанию; это значение может быть представлено как простым, так и сложным типом данных в зависимости от типа свойства.
- **Приоритет значения.** Есть множество механизмов влияния на реальное значение зависимого свойства, в том числе привязка данных, определения стилей, XAML-литералы и анимации.
- **Наследование.** Дочерние элементы могут наследовать значения зависимых свойств от родительских элементов.

Все эти качества делают зависимые свойства идеальным инструментом для встроенной в XAML системы привязки данных.

Присоединенные свойства

Присоединенные свойства — это особый вид зависимых свойств. Они не принадлежат конкретному зависимому объекту, а присоединяются к существующим объектам. Есть два основных довода в пользу применения присоединенных свойств. Первый заключается в том, что через присоединенное свойство можно добавлять к элементу дополнительные атрибуты, не характерные для этого элемента, но полезные в контексте других элементов. В качестве примера можно привести присоединенное свойство `Grid.Row` (строка сетки). Сам по себе элемент для вывода текста `TextBlock` не нуждается в подобном свойстве, так как он может располагаться в различных элементах-контейнерах. Но когда этот элемент находится внутри сетки, возможность точного указания места его расположения может быть полезна, и присоединенное свойство предоставляет сведения подобного рода. В результате родительский элемент `Grid` может узнать его значение для того, чтобы соответствующим образом расположить дочерний элемент.

Запрос значения присоединенного свойства выполняется методом `GetValue`. При вызове этого метода ему нужно передать имя класса, который владеет присоединенным свойством, и имя самого свойства. Например, чтобы узнать, в какой строке сетки находится элемент управления `TextBlock` с именем `ValueText`, можно воспользоваться следующим кодом:

```
var row = ValueText.GetValue(Grid.RowProperty);
```

Второй довод в пользу применения присоединенных свойств заключается в том, что они позволяют задавать многократно используемые режимы работы. Через присоединенные к элементу свойства можно манипулировать этим элементом в коде. Например, представьте, что вам нужно, чтобы кнопка автоматически открывала системное диалоговое окно для передачи данных в другое приложение, помогая пользователю упростить выполнение данной операции. Чтобы не добавлять код на каждую страницу, можно задать для кнопки простой атрибут. При истинном (`True`) значении атрибута кнопка, при щелчке на ней, автоматически откроет диалоговое окно для передачи данных.

Подобный подход реализован в классе `MagicButton` проекта DependencyProperties. Обратите внимание, что данный класс не является наследником класса `DependencyObject`. Присоединенное свойство задается почти так же, как и зависимое свойство, но с использованием другого метода:

```
public static readonly DependencyProperty IsShareButtonProperty =
  DependencyProperty.RegisterAttached(
  "IsShareButton",
  typeof(Boolean),
  typeof(MagicButton),
  new PropertyMetadata(false,
    new PropertyChangedCallback(Changed)));
```

Когда свойство элемента меняется, его владелец, прежде всего, проверяет, является ли он элементом типа `Button` (кнопкой). Если да, событие `Click` перехватывается и используется для вывода диалогового окна:

```
static void button_Click(object sender, RoutedEventArgs e)
{
  DataTransferManager.ShowShareUI();
}
```

И наконец, присоединенное свойство должно предоставлять проекцию, чтобы с ним можно было работать в XAML-коде. Формат проекции немного отличается от проекции зависимых свойств. В случае зависимого

свойства контекст представлен экземпляром зависимого объекта, а в случае присоединенного свойства мы имеем дело со статической ссылкой. Вместо предоставления свойства предусмотрено использование двух статических методов. Они автоматически выделяются из XAML-кода при синтаксическом разборе и модифицируются, чтобы выглядеть в конструкторе как «настоящие» свойства:

```
public static void SetIsShareButton(UIElement element, Boolean
value)
{
  element.SetValue(IsShareButtonProperty, value);
}
public static Boolean GetIsShareButton(UIElement element)
{
  return (Boolean)element.GetValue(IsShareButtonProperty);
}
```

В конструкторе нет команд для методов `Get` и `Set`, поэтому свойство можно присоединить следующим образом:

```
<Button Content="No Magic"/>
<Button Content="Magic"
  local:MagicButton.IsShareButton="True"
  Margin="20 0 0 0"/>
```

После запуска проекта первая кнопка не выполняет никаких функций. Щелчок на второй кнопке приведет к открытию диалогового окна общего доступа. Это действие сейчас ничего не меняет, так как у приложения нет программного кода для обслуживания контракта общего доступа. Больше об этом вы узнаете в главе 8.

Хотя здесь мы рассмотрели очень простой пример, мощь и гибкость присоединенных свойств должна быть для вас совершенно очевидной. Если вы обнаружите, что сложные режимы работы, которые вы разрабатываете, повторяются у различных элементов в системе, рассмотрите возможность соответствующей доработки кода, чтобы сместить акценты в сторону многократно используемых присоединенных свойств. Еще одно преимущество присоединенных свойств заключается в том, что их значения доступны конструктору, что упрощает их применение в XAML другими разработчиками.

Напомним, что класс `DependencyObject` является базовым для системы зависимых свойств. Эта система обеспечивает возможность вычисления значений особых расширенных свойств и предоставляет уведомления при изменении их значений. Это также базовый класс для большинства эле-

ментов управления. Наследование от этого класса позволяет элементам управления представлять их свойства для привязки данных и анимации посредством раскадровки (об этом мы поговорим далее в этой главе).

Привязка данных

Привязка данных подразумевает наличие некоего механизма синхронизации информации между объектом-источником и объектом-приемником (целевым объектом). Источник — это некое хранилище данных, представленное любым CLR-объектом, начиная от простых старых CLR-объектов (Plain Old CLR Object, POCO) и заканчивая зависимыми свойствами или другими сложными типами данных. Источник, кроме того, может быть WinRT-объектом, который имеет атрибут `BindableAttrribute` или реализует интерфейс `ICustomPropertyProvider` (эта книга посвящена разработке программ на C#, но если вы создаете компоненты с использованием C++, вам нужно разобраться в этих двух подходах представления объектов, если хотите, чтобы они участвовали в операциях привязки данных). Целевой объект привязки должен быть свойством `DependencyPropperty` элемента `FrameworkElement`, значение которого синхронизируется с источником данных. Например, это может быть свойство `TextProperty` элемента `TextBox` или `TextBlock`. Объект привязки данных часто ссылается на объект, который реализует интерфейс `IValueConverter`. Подобный объект служит для преобразования типов данных источника и приемника (о конвертерах значений вы узнаете далее в этой главе). На рис. 3.2 показано, как объект привязки данных управляет взаимодействием источника и приемника.

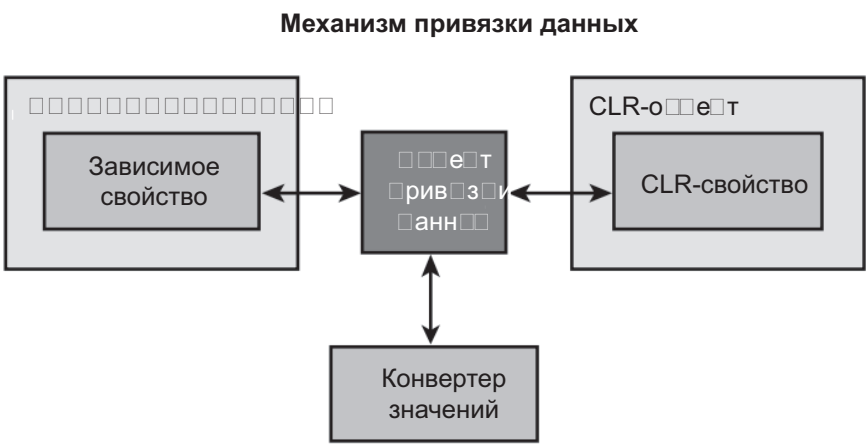


Рис. 3.2. Система привязки данных

Привязка данных задается декларативно с использованием расширенного синтаксиса разметки и ключевого слова `{Binding}`. Вот код разметки, взятый из примера DependencyProperty, который реализует простую привязку данных:

```
<TextBlock Text="{Binding MyNumber}"/>
```

Привязка данных всегда производится к объекту, который задан в качестве контекста данных (`DataContext`), если он не переопределен. Следующий параметр задает путь к свойству в этом объекте, в данном случае — это свойство `MyNumber`. Вы можете передавать объекту `Binding` различные параметры (табл. 3.1).

Таблица 3.1. Параметры привязки данных

Имя параметра	Возможные значения	Описание
Converter	Любой ресурс с ключом, который реализует интерфейс IValueConverter	Позволяет осуществлять преобразование типов данных
Converter-Language	Строка, содержащая название языка	По умолчанию конвертеру передается язык, который определяется контекстом приложения. Если данный параметр установлен, он переопределяет значение, заданное по умолчанию, и преобразователю передается вновь заданное значение
Converter-Parameter	Объект любого типа или ссылка на существующий объект	Объект, заданный в данном параметре, передается конвертеру. Обычно он используется для какой-либо настройки преобразования. Например, передача некоего флага может инвертировать результат логической операции
ElementName	Имя элемента (значение свойства Name) в текущей XAML-области видимости имен (или в области видимости имен шаблонного родителя, если цель привязки находится в шаблоне)	Позволяет осуществлять привязку к другому элементу. Данное свойство является взаимоисключающим для свойств Source и RelativeSource

Имя параметра	Возможные значения	Описание
Mode	OneWay (односторонняя привязка) — обновление целевого свойства происходит в момент привязки, а также тогда, когда меняется контент источника данных. OneTime (единовременная привязка) — обновление целевого свойства осуществляется только в момент привязки. TwoWay (двусторонняя привязка) — обновление целевого свойства происходит в момент привязки, затем изменения в источнике передаются приемнику данных и наоборот	Режим, который определяет направление привязки данных. Для того чтобы решить, какой именно режим нужен в каждом конкретном случае, следует принять во внимание особенности работы реализуемых механизмов. А именно: нужно ли осуществить привязку только для чтения данных из источника или же есть необходимость принимать данные, которые вводит пользователь в целевой элемент интерфейса, и передавать их в источник данных
Path	Путь к свойству объекта-источника данных. Это может быть имя свойства (включая запись через точку для вложенных свойств) или индексатор (то есть запись вида [3].Name для имени свойства третьего элемента в списке и [Foo] для значения записи "Foo" в словаре)	Путь указывает на свойство объекта-источника, которое используется для привязки данных
RelativeSource	TemplatedParent — для элемента управления, где применяется шаблон ControlTemplate. Self — для привязки к собственным свойствам целевого элемента	Это свойство служит для задания расположения источника относительно целевого объекта привязки. Привязку можно осуществить либо к другому элементу управления, либо к самому целевому объекту привязки (например, чтобы при изменении одного свойства менялось другое). Данное свойство и свойства ElementName и Source являются взаимоисключающими

продолжение ⇩

Таблица 3.1 *(продолжение)*

Имя параметра	Возможные значения	Описание
Source	Объект-источник привязки данных, обычно это статический ресурс, заданный в словаре ресурсов (если оставить это значение не заполненным, по умолчанию привязка будет осуществляться к текущему контексту данных (DataContext))	Используйте это свойство для того, чтобы задать объект-источник привязки, который отличается от текущего контекста данных (DataContext) целевого объекта. Часто это либо свойство текущего контекста данных, либо статический ресурс, объявленный в словаре ресурсов

Вы можете найти примеры привязки данных различных видов в проекте DependencyProperties. Вот, например, как реализована непосредственная привязка к элементу управления `Slider` с использованием параметра `ElementName`:

```
<TextBlock Text="{Binding ElementName=Slider, Path=Value}"/>
```

А вот вариант привязки элемента к самому себе, когда в текстовое поле выводится размер шрифта для этого же поля:

```
<TextBlock Text="{Binding RelativeSource={RelativeSource Self},
↳ Path=FontSize}"/>
```

Для того чтобы привязка данных работала правильно, нужно, чтобы объект-источник сообщал системе привязки данных об изменении значений своих свойств. Без подобных уведомлений подсистема привязки данных не сможет обеспечить передачу самых последних значений свойств. Есть два способа реализации отправки подобных уведомлений в коде на C#. Первый заключается в использовании зависимых свойств. Зависимое свойство автоматически принимает участие в работе системы привязки данных и предоставляет необходимые уведомления об изменении свойства.

Второй способ заключается в реализации интерфейса `INotifyPropertyChanged`. Этот интерфейс определяет одно событие, которое называется `PropertyChanged`. Для любого типа привязки, за исключением единовременной (`OneTime`), объект `Binding` подписывается на это событие и наблюдает за изменениями свойств. При использовании данного подхода за вызов события при изменении значений свойств отвечает разработчик. Откройте проект DataBinding и взгляните на класс `DataBindingHost`, представленный в листинге 3.2. Обратите внимание, что класс отслеживает значение свойства `IsOn`, используя закрытое поле. Общедоступное свойство при его

изменении обновляет это поле и вызывает событие PropertyChanged. Для этого служит вспомогательный метод RaisePropertyChanged. Хотя в этом небольшом проекте в подобном методе нет особой необходимости, он может очень пригодиться в более сложных проектах для поддержки уведомлений от нескольких разных свойств.

Листинг 3.2. Уведомление об изменении свойства

```
public class DataBindingHost : INotifyPropertyChanged
{
  private bool _isOn;

  public bool IsOn
  {
    get
    {
      return _isOn;
    }
    set
    {
      _isOn = value;
      RaisePropertyChanged();
    }
  }

  protected void RaisePropertyChanged([CallerMemberName]
    string caller = "")
  {
    PropertyChanged(this, new PropertyChangedEventArgs(caller));
  }

  public event PropertyChangedEventHandler PropertyChanged =
    delegate { };
}
```

Вспомогательный метод использует особый технический прием для вызова события PropertyChanged. Если вы разрабатывали программы для Silverlight или WPF, вы, возможно, вспомните различные трюки, к которым прибегают разработчики для вызова событий. Среди них были, например, передача в метод заранее заданной в коде строки (этот подход мог приводить к ошибкам, если разработчик случайно указывал неверное имя свойства) или передача лямбда-выражения с последующим обходом дерева выражения для извлечения имени свойства (такой подход подразумевал строгую типизацию, но код для его реализации имел неудобный синтаксис). В C# версии 5 можно просто пометить строковый параметр атрибутом CallerMemberName, чтобы во время компиляции вместо него подставлялся

вызываемый метод. В случае со свойством он позволяет задавать имя свойства, что удобно при вызове уведомления об изменении значения свойства.

У вас может появиться вопрос: «Зачем реализовывать этот интерфейс, если зависимые объекты делают то же самое?» Ответ очевиден. Класс `DependencyObject` — это часть инфраструктуры представления данных в XAML. Он расположен в пространстве имен `Windows.UI.Xaml` и его использование имеет смысл при создании нестандартных элементов управления или объектов, которые представляют свойства, нуждающиеся в анимации. Интерфейс `INotifyPropertyChanged` — это часть базовой системной инфраструктуры, он не зависит ни от XAML, ни от UI. Это позволяет создавать облегченные объекты, которые еще называют простыми старыми CLR-объектами (Plain Old CLR Object, POCO). POCO-объекты можно расширять, тестировать и даже использовать в коде, написанном для систем, которые не используют XAML. Данный интерфейс пригоден для отправки уведомлений об изменении свойств везде, где угодно, а объект типа `DependencyObject` должен работать в пределах системы зависимых свойств.

В листинге 3.2 вы могли заметить, как обработчик события устанавливается на пустой делегат. Это помогает упростить код. В редких случаях, когда до вызова обработчика событие не зарегистрировано, подобный код может вбросить исключение. Обычно, однако, перед вызовом обработчика проводят соответствующую проверку. Использование пустого делегата позволяет гарантировать, что как минимум одно событие всегда зарегистрировано, даже если это предлагаемый по умолчанию обработчик, который не выполняет никаких действий.

Главное преимущество привязки данных заключается в четком разделении пользовательского интерфейса и данных. Простой пример — это переменная логического типа. Если она принимает значение `true` (истина), вам нужно показать это элементом интерфейса зеленого цвета, если `false` — красного. Модели данных не нужно работать с цветами, так как это задача пользовательского интерфейса. Вы можете просто задействовать привязку данных для получения необходимого значения и конвертер данных, чтобы преобразовать его к нужному виду.

Конвертеры данных

Откройте папку Converters проекта DataBinding. В ней содержатся три примера конвертеров данных. Все конвертеры данных реализуют интерфейс `IValueConverter`, который определяет два метода: `Convert` и `ConvertBack`. Первый реализуют практически всегда, он применяется при привязке данных для преобразования типа данных из объекта-источника к типу данных, к которому осуществляется привязка в целевом объекте. Второй метод

используют тогда, когда требуется конвертер для реализации двусторонней привязки. Он нужен для выполнения обратной операции — преобразования данных, полученных от целевого объекта, к типу, который используется в источнике. Например, может понадобиться конвертировать текст в элементе управления `TextBox` в число или в значение, соответствующее некоторому цвету, либо привести его к другому типу данных, который требуется в объекте-источнике.

Конвертер видимости преобразует значение логического типа к типу перечисления, которое используется в XAML для управления видимостью элемента:

```
return visible ? Visibility.Visible : Visibility.Collapsed;
```

Если вы привяжете подобное значение напрямую к текстовому свойству, система привязки данных вызовет для этого значения метод `ToString()`, что приведет к передаче имени типа. Используя конвертер, вы можете преобразовать значение в нечто более осмысленное:

```
return truth ?
  "The truth will set you free." :
  "Is falsehood so established?";
```

И наконец, вы можете осуществлять преобразование значений в более сложные объекты, в том числе — в кисти, которые задают различные цвета:

```
return truth ? new SolidColorBrush(Colors.Green)
  : new SolidColorBrush(Colors.Red);
```

Конвертеры хранят в виде ресурсов, которые можно многократно использовать на XAML-страницах. Больше о ресурсах вы узнаете далее в этой главе. Конвертеры в данном примере объявлены с ключами, используя которые можно обращаться к ним в файле MainPage.xaml, например:

```
<converters:VisibilityConverter x:Key="CvtVisibility"/>
```

Здесь показан так называемый *статический ресурс*, так как его нельзя менять, хотя ссылаться на него в XAML-разметке можно несколько раз. Вот как выглядит обращение к статическому ресурсу в коде, который описывает привязку данных (обратите внимание на выделенный курсивом ключ):

```
Visibility="{Binding IsOn,Converter={StaticResource CvtVisibility}}"
```

В листинге 3.3 показан полный набор привязок, используемых в файле MainPage.xaml.

Листинг 3.3. Различные варианты привязки данных

```
<CheckBox IsChecked="{Binding IsOn, Mode=TwoWay}"/>
<TextBlock Text="Is On?" Grid.Column="1"/>
<TextBlock
  Text="Now You See Me."
  Visibility="{Binding IsOn, Converter={StaticResource CvtVisibility}}"
  Grid.Row="1"/>
<TextBlock
  Text="Now You Don't."
   Visibility="{Binding IsOn,Converter={StaticResource CvtVisibility},
    ↳ ConverterParameter=True}"
  Grid.Row="1" Grid.Column="1"/>
<TextBlock Text="{Binding IsOn}" Grid.Row="2"/>
<TextBlock Text="{Binding IsOn, Converter={StaticResource
    ↳ CvtText}}" Grid.Row="2" Grid.Column="1"/>
<Ellipse Grid.Row="3" Grid.ColumnSpan="2" Height="100" Width="100"
  Fill="{Binding IsOn,Converter={StaticResource CvtColor}}"/>
```

Вы можете запустить этот пример, чтобы увидеть, как одни и те же данные могут быть преобразованы в различные интерактивные UI-объекты. Обратите внимание на то, что источник привязки, `DataBindingHost`, определяет единственное значение логического типа (`Boolean`), и он полностью отделен от пользовательского интерфейса. Два различных визуальных представления, которые соответствуют двум состояниям логического значения, показаны на рис. 3.3.

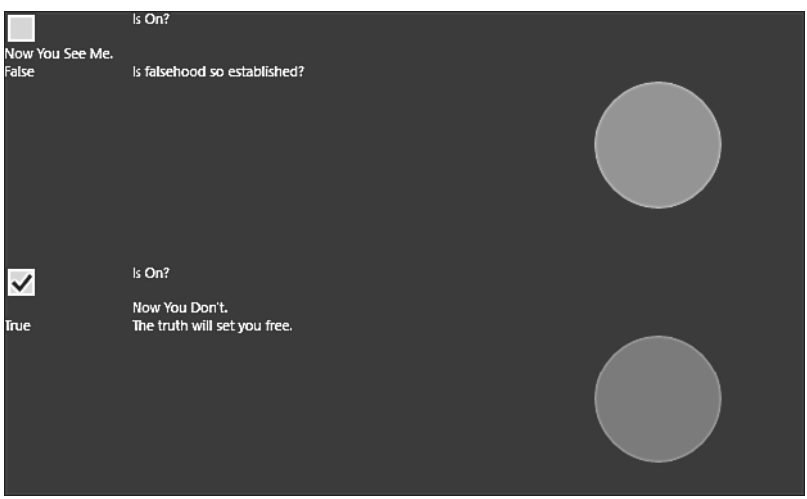


Рис. 3.3. Вот как привязка данных и конвертеры могут изменять внешний вид UI-элементов

Используя привязку данных и конвертеры значений, вы можете преобразовать данные в текст, в цвета и даже управлять с их помощью видимостью элементов.

Раскадровки

Раскадровки (storyboards) воздействуют на зависимые объекты, обновляя их свойства и влияя на целевое значение в течение некоторого времени. Обычно подобный режим работы применяется при анимации изменений состояний объекта, а также для привлечения внимания пользователя к каким-либо событиям. Класс `Storyboard` (раскадровка) является наследником класса `Timeline` (временная шкала). В табл. 3.2 приведены свойства временной шкалы и описаны особенности их применения.

Таблица 3.2. Свойства класса Timeline

Свойство	Описание
AutoReverse	Когда это свойство установлено в значение true (истина), временная шкала начнет воспроизводиться в обратном направлении после завершения воспроизведения в прямом направлении
BeginTime	Время начала анимации определяет задержку между запуском анимации и первым изменением целевого свойства. Например, время начала, равное 2 секундам, отложит запуск анимации на 2 секунды от момента, когда будет вызван метод Begin временной шкалы, который инициирует ее воспроизведение
Duration	Позволяет указать длительность анимации, которая задана временной шкалой
SpeedRatio	Задает коэффициент скорости течения времени объекта по отношению к родительскому объекту. Установка этого значения в 0,5 приведет к тому, что анимация, заданная временной шкалой, будет воспроизводиться на скорости, равной половине скорости родительской временной шкалы
FillBehavior	Это важное свойство указывает, что произойдет при остановке воспроизведения временной шкалы. Если оно установлено в значение Stop (по умолчанию), при завершении воспроизведения временная шкала больше не воздействует на целевое свойство. Если оно установлено в значение HoldEnd, временная шкала продолжает воздействовать на значение целевого свойства даже после завершения анимации (но только до тех пор, пока анимация не будет остановлена явным образом)

продолжение ⇩

Таблица 3.2 *(продолжение)*

Свойство	Описание
Repeat-Behavior	Данное свойство может иметь несколько разных значений. Оно представлено типом данных RepeatBehaviour, целое значение его поля Count определяет количество повторений временной шкалы (включая воспроизведение в обратном порядке, если установлено свойство AutoReverse объекта Timeline). Поле Duration определяет общее время воспроизведения временной шкалы (например, временная шкала с длительностью 30 секунд и значением RepeatBehaviour, установленным на 2 минуты, будет воспроизведена 4 раза). Особое значение Forever позволяет повторять воспроизведение временной шкалы бесконечно
Completed	Это событие вызывается, когда завершается выполнение действий, заданных временной шкалой

Откройте проект Storyboards.

Он иллюстрирует разные варианты применения временной шкалы. Вот пример определения раскадровки из проекта:

```
<Storyboard x:Name="FirstOneAnimation" Storyboard.
TargetName="FirstOne">
  <DoubleAnimation Duration="0:0:5"
    Storyboard.TargetProperty=
      "(Rectangle.RenderTransform).(ScaleTransform.X)"
    From="-300" To="300"/>
</Storyboard>
```

Целевым объектом является прямоугольник, который объявлен в XAML и окрашен в красный цвет благодаря применению к нему соответствующего стиля (о стилях мы поговорим в следующем разделе). Раскадровка может использовать свойство `TargetName` для обращения к элементу по имени или свойство `Target` для обращения к определенному объекту, например к ресурсу.

Свойство `Target` может содержать имя свойства или описание пути к свойству. Описание пути включает в себя каждый объект из XAML-иерархии, упоминание которого заключено в скобки, далее следует точка, затем — описание следующего объекта и его свойства.

Паттерн подобной конструкции выглядит так:

```
(object.property).(object.property)
```

В примере целевым объектом является прямоугольник, поэтому первая часть пути указывает на свойство `RenderTransform` целевого объекта `Rectangle` (прямоугольник). Данное свойство содержит описание матричного преобразования, которое можно использовать для того, чтобы управлять внешним видом элемента. В данном примере применяется трансформация сдвига (translation), благодаря которой прямоугольник перемещается вдоль оси *X*. Точка указывает на то, что путь ведет к значению, назначенному трансформации, которое, в данном случае, является объектом типа `TranslateTransform`. В итоге целевым свойством для анимации является свойство `x` данного объекта.

В пример включены и другие раскадровки, иллюстрирующие различные варианты их использования. Второй пример автоматически повторяет анимацию, сначала выполняя ее в прямом порядке, потом — в обратном, и так — до тех пор, пока анимация не будет остановлена явным образом. Он также включает в себя функцию сглаживания, которая выполняет вычисления для анимации. Анимация основана на кадрах. Когда задана длительность анимации, вычисляется количество кадров, которое необходимо видеокарте, чтобы вывести данную анимацию на экран. При этом определяются изменения, которые должны быть применены к каждому кадру. Вам не нужно беспокоиться о том, насколько быстро осуществляется вывод данных на экран, так как это делается автоматически. Вам лишь нужно задать начальное и конечное значение анимации, а также ее длительность.

Можно создавать и дискретные анимации, которые изменяют значения не плавно, а прерывисто. Третья раскадровка иллюстрирует механизм управления цветом. Вместо плавного перехода между цветами она немедленно изменяет цвет после наступления *времени ключевого кадра*, то есть момента, когда должно измениться дискретное значение. Последняя анимация так же использует дискретное значение, но иллюстрирует пример анимации видимости объекта, управляя значением параметра `Visibility.Collapsed`.

В каждой раскадровке задана «базовая временная шкала», на основе которой работают дочерние раскадровки. В дополнение к базовой временной шкале раскадровка может содержать набор других временных шкал (или других раскадровок и анимаций). Как и ко всем зависимым объектам, к раскадровке применимы правила наследования. Если вы зададите длительность анимации на уровне родительской раскадровки, все дочерние временные шкалы унаследуют это значение. Если для дочернего элемента задана длительность, она переопределит значение, заданное в элементе-родителе.

В платформу встроено четыре вида анимации. Вы можете создавать и собственные анимации, но перечисленных обычно хватает в подавляющем большинстве случаев.

- `ColorAnimation`. Анимация цветового свойства (типа `Color`) для зависимого свойства и перехода от начального цвета к конечному.
- `DoubleAnimation`. Анимируется изменение числового значения удвоенной точности (типа `Double`) в пределах двух целевых значений. Большинство свойств в среде исполнения имеют тип `Double`, поэтому данная анимация применяется во многих случаях.
- `PointAnimation`. Анимация перехода между двумя целевыми точками (типа `Point`); обычно используется для управления геометрическими объектами.
- `ObjectAnimationUsingKeyFrames`. Позволяет задавать дискретные значения в определенные моменты анимации.

Первые три варианта анимации также могут быть заданы в дискретном виде с использованием ключевых кадров. В случае базовой анимации осуществляется плавное изменение значения в заданном диапазоне за указанное время. В случае дискретной анимации значения резко изменяются в заданное время в ключевых кадрах анимации. Например, можно начать анимацию, сделав элемент видимым, переместить его по экрану, плавно увеличить его прозрачность, а в конце, в ключевом кадре, скрыть его, убрав из визуального дерева.

Класс `Storyboard` расширяет методы и события, описанные в табл. 3.2, методами запуска (`Begin`), приостановки (`Pause`), возобновления (`Resume`), завершения (`Stop`) и смещения (`Seek`) раскадровки на заданное время. Вот их краткое описание:

- `Begin` — запуск анимации.
- `Stop` — завершение анимации.
- `Pause` — приостановка воспроизведения анимации с сохранением сведений о текущей позиции на шкале времени.
- `Resume` — воспроизведение анимации, начиная с позиции, в которой она была приостановлена.
- `Seek` — перемещение по шкале времени для запуска анимации с заданного момента.

В примере раскадровки иллюстрируется несколько простых анимаций. Но анимации становятся интереснее с трансформациями, в ходе которых элементы можно сдвигать, вращать, растягивать, сжимать и даже имитировать их перемещение в трехмерном пространстве с использованием проекций.

Анимации можно также изменять с помощью специальных функций сглаживания. По умолчанию временная шкала линейна, и значения меняются

с постоянной скоростью. Функция сглаживания меняет скорость анимации, позволяя добиваться более естественного поведения объектов. В платформу встроено множество таких функций. Например, вот описания двух из них:

- Сглаживание отскока (класс `BounceEasy`) позволяет имитировать постепенно ускоряющееся изменение значения в заданных пределах, что приводит к эффекту, похожему на поведение баскетбольного мяча, который скачет по паркету.
- Нелинейное сглаживание (класс `CircleEase`) позволяет быстро ускорять или замедлять движение элемента. Благодаря такому сглаживанию, объект после быстрого перемещения при достижении целевого значения может резко замедлиться.

Возможность отделить режимы работы пользовательского интерфейса, в котором применяются раскадровки, от функциональности приложения, очень важна. Дизайн — это тоже важный аспект приложений, но он не является основной темой данной книги. К счастью, я, как разработчик, могу абстрагироваться от пользовательского интерфейса, работая над кодом. Когда пользователь выбирает пункт меню, я могу запрограммировать поведение системы в ответ на это действие, в то время как дизайнер занимается обработкой UI-событий, созданием анимаций и, при необходимости, других эффектов. Подробнее об этой абстракции мы поговорим позже. Раскадровки — это мощный инструмент, с помощью которого можно добавить в приложение вспомогательные элементы, помогающие пользователю понять, когда на некоторый элемент наведен фокус ввода, когда нужно что-то сделать, и даже для вывода всплывающих уведомлений, о чем-либо сообщающих пользователю. С раскадровками работает диспетчер визуальных состояний (Visual State Manager, VSM), предлагая некоторые полезные сервисы, которые мы рассмотрим далее в этой главе.

Стили и ресурсы

В предыдущих примерах вы могли заметить, что элемент `Grid` содержит коллекцию `Resources` (ресурсы), в которой есть элемент `Style` (стиль). Стили — это особый вид зависимых объектов. Содержимое объекта `Style` представлено списком элементов `Setter` (установщики значений). Установщики значений предоставляют значения для свойств. Исключительная полезность стилей определяется тем, что стиль можно определить один раз, после чего применять его ко множеству элементов. Стили могут неявно воздействовать на элементы заданного типа, они могут быть основаны на других стилях, образуя иерархическую структуру.

Правильное описание и применение стилей — это ключ к успеху в разработке приложений для Windows 8, так как они позволяют отделить особенности внешнего вида и режима работы элементов от их описаний. Разработчик может не беспокоиться о таких деталях, как размер шрифта или параметры цветового градиента, если эти параметры могут быть заданы независимо с помощью стилей. Стиль определяется в классе `FrameworkElement`, что позволяет применять стили к любым элементам, которые являются потомками этого класса.

Вспомните, что класс `DependencyObject` является базовым для системы зависимых свойств. Его возможности расширяет класс `UIElement`, который является базовым для объектов, имеющих визуальное представление и поддерживающих обработку ввода. В классе `UIElement` объявляются события, связанные с клавиатурой, указателями, касаниями, манипуляциями и фокусом ввода. Возможности данного класса расширяет класс `FrameworkElement`. Этот класс определяет свойства, необходимые для построения макетов, события жизненного цикла элемента, такие как `Loaded` (загрузка) и `Unloaded` (выгрузка). Благодаря ему реализуются привязка данных и поддержка стилей.

Стили чаще всего объявляют в виде ресурсов. В XAML UI-элементы могут содержать коллекцию ресурсов, которыми могут пользоваться как сами эти элементы, так и их потомки. Ресурсы — это обычные словари, содержащие многократно используемые значения. Каждое значение имеет ключ, по которому к нему можно обращаться. Пример использования ресурсов включает в себя раскадровки, стили и конвертеры значений. Основное ограничение ресурсов заключается в том, что они не могут быть элементами визуального дерева. Причиной данного ограничения является то, что любой элемент может присутствовать в визуальном дереве лишь в единственном экземпляре, ресурсы же могут использоваться многократно, и это может привести к тому, что на них будут ссылаться несколько элементов. Подобная ситуация, если говорить о визуальном дереве, приведет к ошибке, так как элемент, которым мог бы быть стиль, уже является потомком другого элемента.

В XAML ресурсы располагаются в области видимости UI-элемента и наследуются, о чем вам, возможно, уже известно. Ресурсы, объявленные на уровне страницы (элемент `Page`), доступны для всех элементов страницы, в то время как ресурсы, заданные для конкретного элемента `Grid` (сетка), доступны лишь его дочерним элементам. Кроме того, ресурсы для упрощения доступа к ним можно группировать в коллекции, которые называют *словарями ресурсов*. В проекте StoryboardExamples откройте файл App.xaml. Обратите внимание, что в нем имеется тег `ResourceDictionary`. Внутри описания словаря есть ссылка на файл со стилями:

```
<ResourceDictionary Source="Common/StandardStyles.xaml"/>
```

Ссылка на словарь расположена на верхнем уровне иерархии приложения, поэтому ресурсы доступны из любой его части. В результате кнопка и текст имеют характерный стиль Windows 8, даже если для них не заданы особые атрибуты. Если вы откроете файл StandardStyles.xaml, на который сделана ссылка, то увидите стандартные стили, применяемые в приложениях для Windows 8. Воспринимайте эти стили как образец, пользуйтесь ими в собственных приложениях. Единообразная стилизация важна для того, чтобы приложение выглядело как органичная часть системы и пользователю было удобно с ним работать. В файле вы обнаружите определения стилей для кнопок, стандартных значков и даже для наиболее распространенных анимаций и переходов.

Доступ к ресурсам можно получить и из кода. Коллекция ресурсов — это словарь. В примере главная сетка приложения содержит раскадровку. Вы можете получить к ней доступ следующим образом:

```
var firstOne = (Storyboard)MainGrid.Resources["FirstOneAnimation"];
```

Когда вы щелкаете на кнопке, которая расположена на странице приложения, обработчик событий перебирает список ресурсов в главной сетке, находит ресурсы, имеющие тип `Storyboard`, и вызывает их методы `Begin`. Это автоматически запускает все анимации, заданные с помощью раскадровок. Если вы добавите в коллекцию еще одну раскадровку, она также будет обработана при щелчке на кнопке:

```
foreach (var storyboard in
  MainGrid.Resources.Values.OfType<Storyboard>())
{
  storyboard.Begin();
}
```

Некоторые стили, хотя они и определены в виде ресурсов, влияют на элементы управления, даже если вы не задавали их стили. Это так называемые скрытые стили. Скрытые стили автоматически применяются к любым экземплярам класса в пределах текущей области видимости. Для таких стилей ключ для явной ссылки на стиль не требуется, они ориентированы на типы и применяются неявно. Например, цвет элементов типа `Rectangle` задан с помощью скрытого стиля:

```
<Style TargetType="Rectangle">
  <Setter Property="Fill" Value="Red"/>
</Style>
```

У явных стилей есть связанные с ними ключи. Эти ключи нужно указывать при объявлении элементов, которые будут использовать данные стили. Фон в приложении для Windows 8 задан путем явного указания стиля:

```
<Grid
  x:Name="MainGrid"
  Background="{StaticResource ApplicationPageBackgroundThemeBrush}">
```

Экземпляры класса `Style` поддерживают сложные вложенные объекты и присоединенные свойства. Они могут быть целевыми атрибутами, вложенными свойствами и определять сложные объекты, такие как кисти. Всякий раз, когда вы обнаруживаете, что занимаетесь настройкой полей или форматированием, вам следует остановиться и сделать то же самое посредством стилей. Благодаря стилям, все параметры хранятся в одном и том же месте, что позволяет последовательно использовать их во всем приложении и упрощает их изменение. Многие примеры в этой книге предполагают неявное форматирование, что упрощает их структуру. В последних примерах с помощью стилей определяются внешний вид и режимы работы приложения.

Экземпляры класса `Style` воздействуют на цветовую тему приложения. Этот пример иллюстрирует объявления стилей, внедренные непосредственно в раздел ресурсов элемента управления. В приложениях, однако, чаще выделяют специальную папку, в которой располагаются файлы определения стилей. Если нужно применить стили, ссылаются на нужные файлы в этой папке. Шаблоны приложений, написанных на C# и XAML, автоматически предоставляют папку для определения стилей.

Макет

Приложения для Windows 8 должны обладать гибким и реактивным макетом, который хорошо работает на экране любого форм-фактора. В частности, это касается портретной и альбомной ориентации экрана, различных размеров экрана, а также режима фиксации. XAML предоставляет множество UI-элементов, призванных упростить построение макета для вашего приложения.

Стандартное приложение состоит из *визуального корня*, или «главного» UI-элемента, который представляет собой всю рабочую поверхность приложения. Обычно это элемент типа `Frame` (рамка) — особый контейнер, который поддерживает навигацию по страницам. Контейнер может менять отображаемые в нем элементы `Page` (страницы), которые являются логически-

ми страницами приложения. Больше о подсистеме навигации вы узнаете в главе 5.

Внутри каждой логической страницы есть набор элементов управления, на основе которых и строится макет. Они обеспечивают форматирование и задают взаимное расположение элементов, которые вы хотите показывать на странице. Существует немало элементов управления для создания макетов: от простых, описывающих поверхности для расположения других элементов, до элементов управления списками и особых новых наборов элементов управления, которые поддерживают группирование данных. О них вы узнаете позже, а для начала мы рассмотрим несколько простых панелей, пришедших к нам из платформ Silverlight и WPF. Примеры большинства из этих панелей вы можете найти в проекте Layout для главы 4.

Элемент управления Canvas

Элемент управления `Canvas` (холст) отлично подходит для создания рабочих поверхностей фиксированного размера. Он используется, когда нужна особая точность расположения элементов в контейнере, а гибкость макету не требуется. Холст организует другие элементы управления путем задания их координат на плоскости. Его дочерние элементы располагаются в соответствии со смещением, которое задают от левого верхнего угла холста до левого верхнего угла элемента. Элемент управления `Canvas`, кроме того, позволяет задать значение `ZIndex`, которое управляет выводом объектов с учетом их взаимного расположения. Чем большее значение `ZIndex` соответствует некоторому элементу, тем «выше» по отношению к другим элементам он располагается.

Холст всегда служит для создания фиксированного макета. Он не реагирует на события изменения размера и соответственно не перераспределяет дочерние элементы, чтобы они лучше размещались на экранах различных размеров. Это наименее гибкий из всех встроенных элементов управления, представляющих панели для размещения других элементов.

Вот пример кода, который описывает элемент управления `Canvas` и его контент:

```
<Canvas Width="400" Height="400" Background="White">
  <Rectangle Fill="Red" Width="200" Height="150"/>
  <Rectangle Fill="Green" Width="150" Height="75"/>
  <Ellipse Fill="Blue" Width="250" Height="250"
  Canvas.Left="75" Canvas.Top="75" Canvas.ZIndex="-1"/>
</Canvas>
```

Результат визуализации этого кода на странице приложения приведен на рис. 3.4.

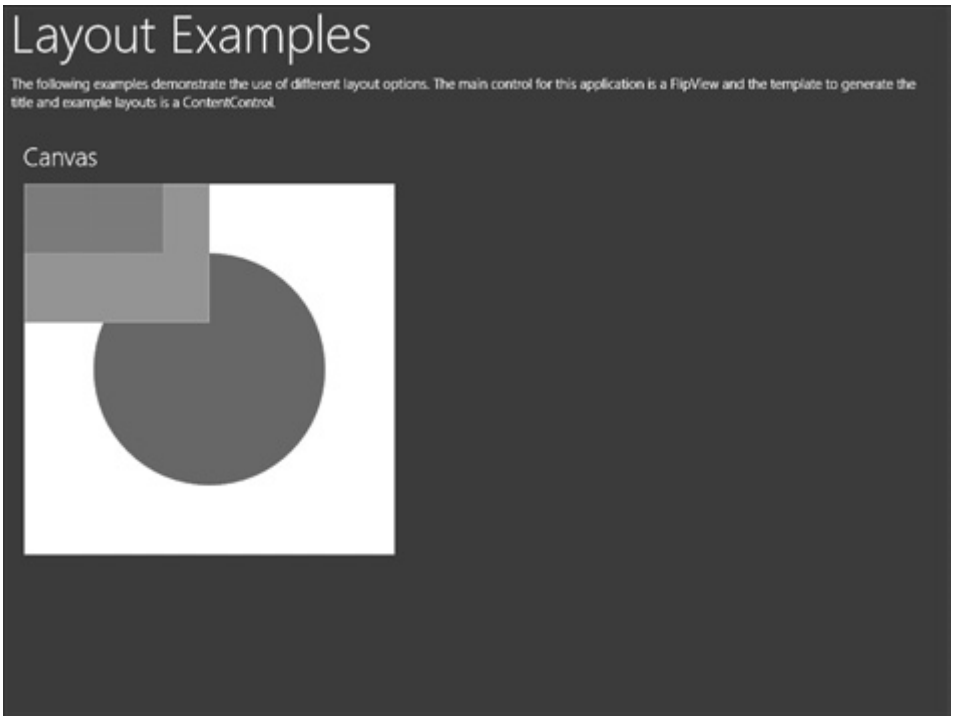


Рис. 3.4. Макет, построенный на основе элемента управления Canvas

Элемент управления Grid

На сегодняшний день `Grid` (сетка) является самым мощным и самым распространенным элементом управления для создания макетов. Кроме того, сетка по умолчанию используется в макетах приложений в Visual Studio. Сетку часто сравнивают с HTML-таблицей, так как она позволяет задавать строки и столбцы. Система макетов в XAML отличается большой гибкостью. Вы можете задать размер ячеек сетки в пикселах, можно также сделать так, чтобы размер автоматически подстраивался под содержимое ячейки или относительно других ячеек. Обычный подход к построению макета заключается в том, чтобы задать автоматически настраиваемую или фиксированную высоту или ширину навигационной панели или ленты инструментов, а затем оставшимся ячейкам позволить автоматически принять размер, соответствующий доступному свободному месту.

Одна из важных особенностей элемента `Grid` заключается в том, что он отлично подходит для создания гибких макетов. Сетка может растягиваться,

чтобы заполнять доступное место в контейнере родительского элемента, а также менять размер при изменении размера ее содержимого. Есть три варианта установки свойства `GridLength`, которое определяет ширину столбцов сетки и высоту ее строк:

- Задание размера в пикселах.
- Указание параметра `Auto`, чтобы задать размер ячейки, основываясь на размерах дочернего элемента, который в ней расположен.
- Нотация со звездочками позволяет задать долю доступного свободного места, которую должна занять ячейка.

Нотация со звездочками, вероятно, сложнее всего для понимания. Модификатор в виде символа звездочки просто указывает на оставшееся свободное пространство. Сама по себе звездочка представляет одну единицу незаполненной области. Если вы указываете вместе с ней некоторое значение, произойдет соответствующее изменение пропорций. В табл. 3.3 показано, как вычисляются итоговые значения размеров, заданных с использованием нотации со звездочками, в сетке размером 400 пикселов в ширину.

Таблица 3.3. Размеры столбцов с применением нотации со звездочками для сетки шириной 400 пикселов

Столбец 1	Столбец 2	Столбец 3	Формула
*	*	*	1+1+1=3. Каждый столбец займет 1/3 от 400
133,33	133,33	133,33	
*	0.5*	0.5*	1+0,5+0,5=2. Первый столбец, таким образом, будет иметь ширину, равную 1/2 от 400. Оставшиеся столбцы получат по 1/4 части 400 (0,5/2)
200	100	100	
1*	2*	3*	1+2+3=6. Первый столбец займет 1/6 часть от 400, второй — 1/3, третий — 1/2
66,666	133,334	200	
100	0.5*	2*	0.5+2=2.5. Первый столбец занимает 100, двум другим остается 300. Второй столбец, таким образом, получает 0,5/2,5=1/5 от 300, третий — 2/2,5=4/5
100	60	240	

Важно понимать, как сетка обеспечивает размещение элементов и задает их размеры, так как это может повлиять на то, как вы будете разрабатывать дизайн нестандартных элементов управления. В листинге 3.4 показан простой макет на основе сетки.

Листинг 3.4. XAML-разметка макета на основе элемента управления Grid

```
<Grid Background="White" Width="400" Height="400">
  <Grid.RowDefinitions>
    <RowDefinition Height="1*"/>
    <RowDefinition Height="2*"/>
  </Grid.RowDefinitions>
  <Grid.ColumnDefinitions>
    <ColumnDefinition Width="1*"/>
    <ColumnDefinition Width="2*"/>
  </Grid.ColumnDefinitions>
  <Rectangle Fill="Red" Width="250" Height="125"/>
  <Rectangle Fill="Green" Width="150" Height="75"
    Grid.Row="1"/>
  <Ellipse Fill="Blue" Width="250" Height="250"
    Grid.Row="1" Grid.Column="2"/>
</Grid>
```

Результат визуализации этой разметки приведен на рис. 3.5.

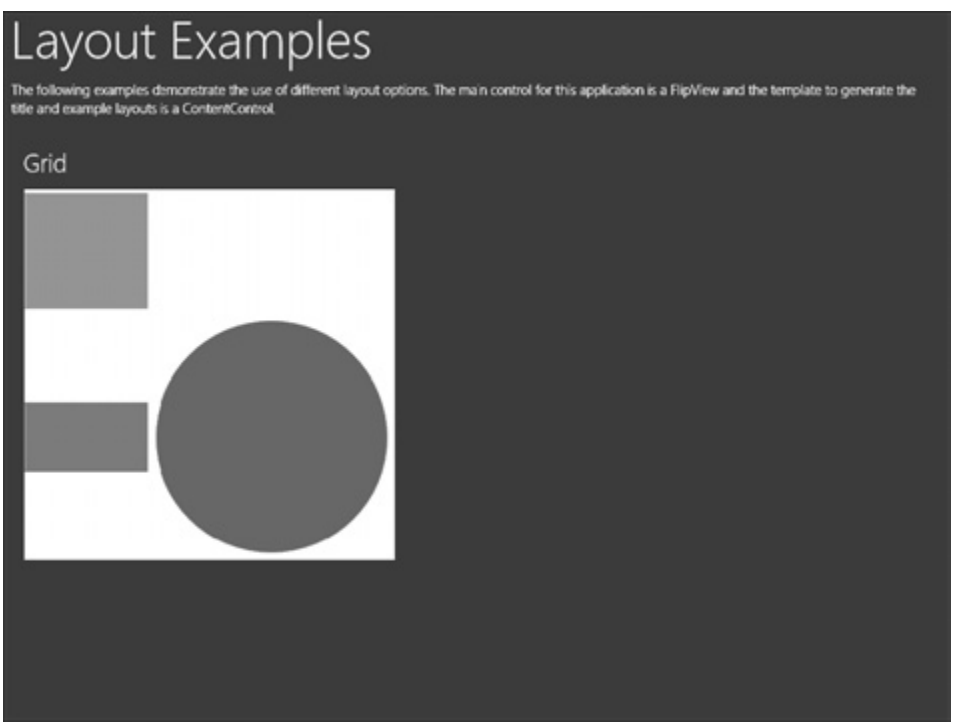


Рис. 3.5. Макет на основе элемента управления Grid

Элемент управления StackPanel

Элемент управления `StackPanel` (стек-панель) — это панель, у которой, в зависимости от ее ориентации, вложенные элементы располагаются либо горизонтально, в ряд, либо вертикально, один над другим. Она хорошо подходит, когда вам не нужно размещать элементы по строкам или столбцам или когда у вас есть динамический список элементов, которые вы планируете добавлять в панель без необходимости предварительно настраивать столбцы или строки. Основываясь на собственной ориентации, элемент управления `StackPanel` автоматически вычисляет высоту или ширину каждого элемента и располагает его либо правее, либо ниже элемента, уже присутствующего в панели.

Потенциальная проблема при использовании элемента управления `StackPanel` заключается в том, что он при размещении в нем дочерних элементов выделяет им неограниченное пространство. Поэтому для того, чтобы просмотреть все элементы внутри `StackPanel`, может понадобиться элемент управления `ScrollViewer`, обеспечивающий прокрутку контента панели. Если вам нужно разместить вложенные элементы управления в области заданного размера, воспользуйтесь контейнером `WrapGrid`, который описан далее. Стек-панель отлично подходит для размещения небольших наборов элементов, которые гарантированно уместятся в пределах доступного свободного пространства.

Вот пример описания элемента управления `StackPanel`:

```
<StackPanel Background="White" Width="400" Height="400">
  <Rectangle Fill="Red" Width="250" Height="125"/>
  <Rectangle Fill="Green" Width="150" Height="75"/>
  <Ellipse Fill="Blue" Width="250" Height="250"/>
</StackPanel>
```

На рис. 3.6 показано, как это выглядит в окне приложения.

Элементы управления VirtualizingPanel и VirtualizingStackPanel

Специализированная панель, которую обычно называют *виртуализирующей* (virtualizing panel), помогает работать с данными большого объема. Элемент `ListBox` (список) по умолчанию использует виртуализирующую стек-панель в качестве контейнера для размещения контента. Причем

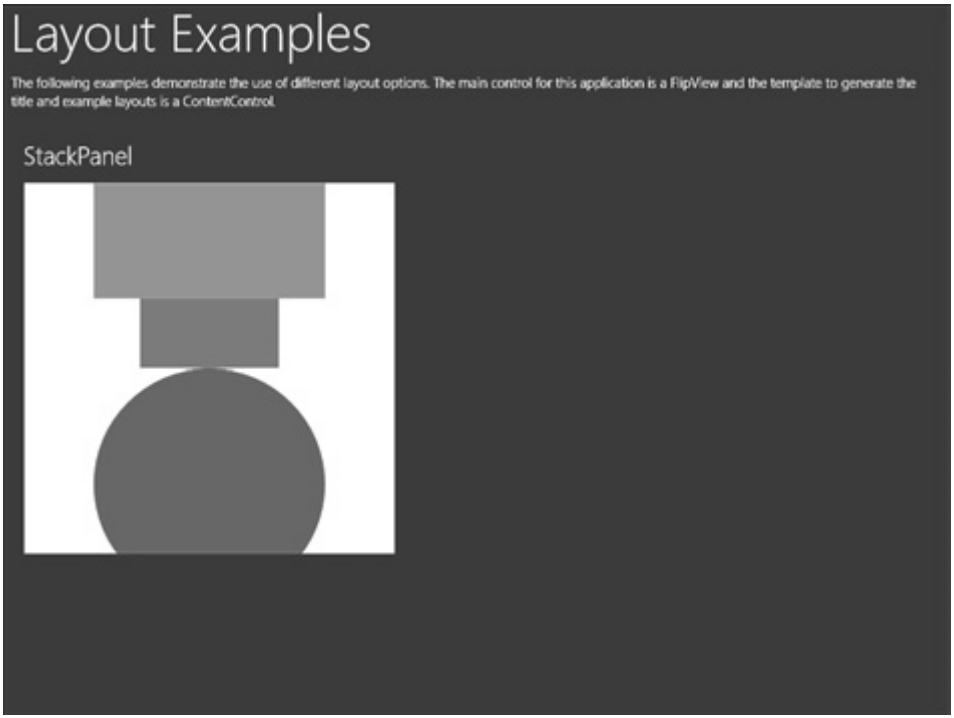


Рис. 3.6. Макет на основе элемента управления StackPanel

этот контейнер в `ListBox` можно заменить каким-нибудь другим, однако в большинстве случаев от него нет смысла отказываться.

Обычная стек-панель вмещает в себя неограниченное количество элементов, так как она предоставляет им неограниченное пространство. Если вы разместите в ней 5000 элементов, она выведет их все, создав соответствующие элементы управления, причем даже в том случае, если видимыми будут только десять. Это ведет к серьезной дополнительной нагрузке на систему при работе с большими наборами данных.

В то же время виртуализирующая стек-панель учитывает размер видимой области и выделяет из полного набора размещенных в ней данных только то подмножество, которое можно вывести на экран за один раз. Если видимыми могут быть лишь десять элементов, то такая стек-панель создаст лишь десять элементов управления для их визуализации. Когда пользователь прокручивает список, в нем сохраняется то же самое количество элементов управления, изменяются лишь содержащиеся в них данные. О том, как работают подобные шаблоны данных, вы узнаете в следующей главе.

Недостаток виртуализирующей стек-панели заключается в том, что прокрутка данных происходит не плавно. В примере, демонстрирующем различные макеты, выведены два списка. В одном из них вместо виртуализирующей стек-панели применяется обычная. Вы обнаружите, что в обычной стек-панели прокрутка выполняется плавно. Вы можете прокрутить список на несколько пикселов, и сверху или снизу списка появится часть элемента, который был до этого не виден. В случае же с виртуализирующей стек-панелью прокрутка возможна лишь с шагом в один элемент. Она не поддерживает прокрутку с частичным выводом элемента, так как в ходе прокрутки осуществляется замена контента видимых элементов. В итоге при использовании обычных и визцализирующих панелей вам придется выбирать между обработкой больших объемов данных без падения производительности системы и более удобным пользовательским интерфейсом. Кроме того, вы можете разработать нестандартный элемент управления, совмещающий в себе достоинства обоих элементов. Подобные решения можно найти среди наборов элементов управления, предоставляемых многими сторонними разработчиками.

Элемент управления WrapGrid

Этот элемент управления представляет собой особый вид сетки, которая поддерживает автоматическое создание строк и столбцов на основе размещаемых в ней данных. В отличие от элемента управления `Grid`, который требует предварительного задания параметров строк и столбцов, `WrapGrid` (сетка с автоматическим переносом контента в следующую строку или столбец) делает это автоматически. Подобная сетка размещает дочерние элементы один за другим, либо слева направо, либо сверху вниз, в зависимости от ориентации. А затем, когда пространство строки или столбца оказывается занятым, переносит следующий элемент в следующую строку или столбец.

Пример этого мощного элемента управления вы можете найти в приложении Chapter3Panels. Он позволяет вложенным элементам управления занять все имеющееся пространство, все его строки и столбцы. Пользователь может воспользоваться прокруткой, чтобы увидеть элементы, которые не поместились в видимой области. Это позволяет вам не заниматься самостоятельным расчетом параметров элемента управления, так как сетка автоматически подстраивается под текущее разрешение экрана.

Элемент `WrapGrid` всегда определяют как часть шаблона для набора элементов управления, как показано в листинге 3.5.

Результат визуализации этой разметки приведен на рис. 3.7.

Листинг 3.5. XAML-разметка макета на основе элемента управления WrapGrid

```
<ItemsControl>
  <ItemsControl.ItemsPanel>
    <ItemsPanelTemplate>
      <WrapGrid
        ItemHeight="200"
        ItemWidth="200"
        MaximumRowsOrColumns="2"
        Width="400"
        Height="400"
        Background="White"/>
    </ItemsPanelTemplate>
  </ItemsControl.ItemsPanel>
  <ItemsControl.Items>
    <Rectangle Fill="Red" Width="250" Height="125"/>
    <Rectangle Fill="Green" Width="150" Height="75"/>
    <Ellipse Fill="Blue" Width="250" Height="250"/>
  </ItemsControl.Items>
</ItemsControl>
```

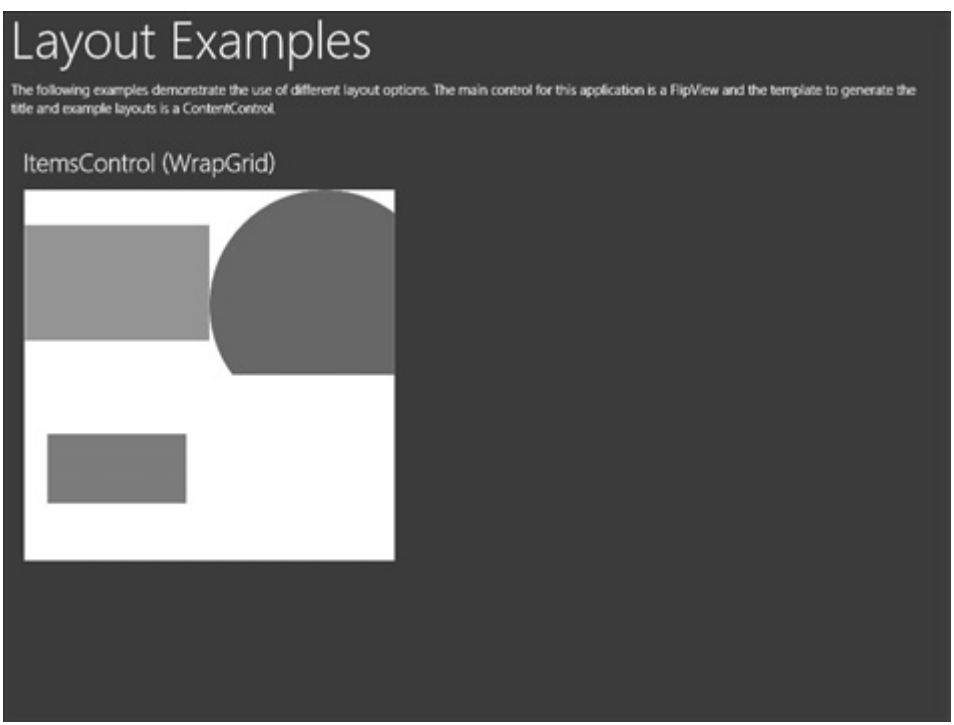


Рис. 3.7. Макет, построенный на основе элемента управления WrapGrid

Элемент управления VariableSizedWrapGrid

Начальный экран Windows 8 — отличный пример возможностей элемента управления `VariableSizedWrapGrid` (сетка с автоматическим переносом контента в следующую строку или столбец с переменным размером ячеек). Обратите внимание на то, что некоторые плитки длиннее, чем другие. Это означает, что элементы в такой сетке могут иметь разный размер. Подобный подход часто используется для вывода с помощью сетки элементов разных типов, когда некоторые из них имеют размеры и ориентацию, отличающиеся от тех же параметров других элементов.

Есть два способа задания количества ячеек под один элемент в сетке. Первый — через присоединенные свойства `VariableSizedWrapGrid.ColumnSpan` и `VariableSizedWrapGrid.RowSpan` дочернего элемента. Они указывают сетке-контейнеру на то, сколько ячеек нужно выделить для размещения элемента.

Если вы привязываете к сетке список элементов, вам понадобится использовать подкласс содержащего их объекта (обычно — `GridView`) и переопределить метод `PrepareContainerForItemsOverride`. Далее в этой главе мы обсудим это подробнее, когда будем рассматривать пример Panels.

В листинге 3.6 приведен пример макета, созданного с использованием данного элемента управления.

Листинг 3.6. XAML-разметка макета на основе элемента управления VariableSizedWrapGrid

```
<ItemsControl>
  <ItemsControl.ItemsPanel>
    <ItemsPanelTemplate>
      <VariableSizedWrapGrid
        MaximumRowsOrColumns="2"
        ItemWidth="200"
        ItemHeight="200"
        Width="400"
        Height="400"
        Background="White"/>
    </ItemsPanelTemplate>
  </ItemsControl.ItemsPanel>
  <ItemsControl.Items>
    <Rectangle Fill="Red" Width="250" Height="125"
    VariableSizedWrapGrid.ColumnSpan="2"/>
    <Rectangle Fill="Green" Width="150" Height="75"/>
    <Ellipse Fill="Blue" Width="250" Height="250"/>
  </ItemsControl.Items>
</ItemsControl>
```

Результат визуализации этого макета приведен на рис. 3.8. Обратит внимание на то, что благодаря различному размеру ячеек и перекрытию первый прямоугольник удалось показать полностью. Окружность, однако, оказалось обрезанной, так как размер видимой области макета в этом примере ограничен величиной 400 пикселов в высоту и ширину.

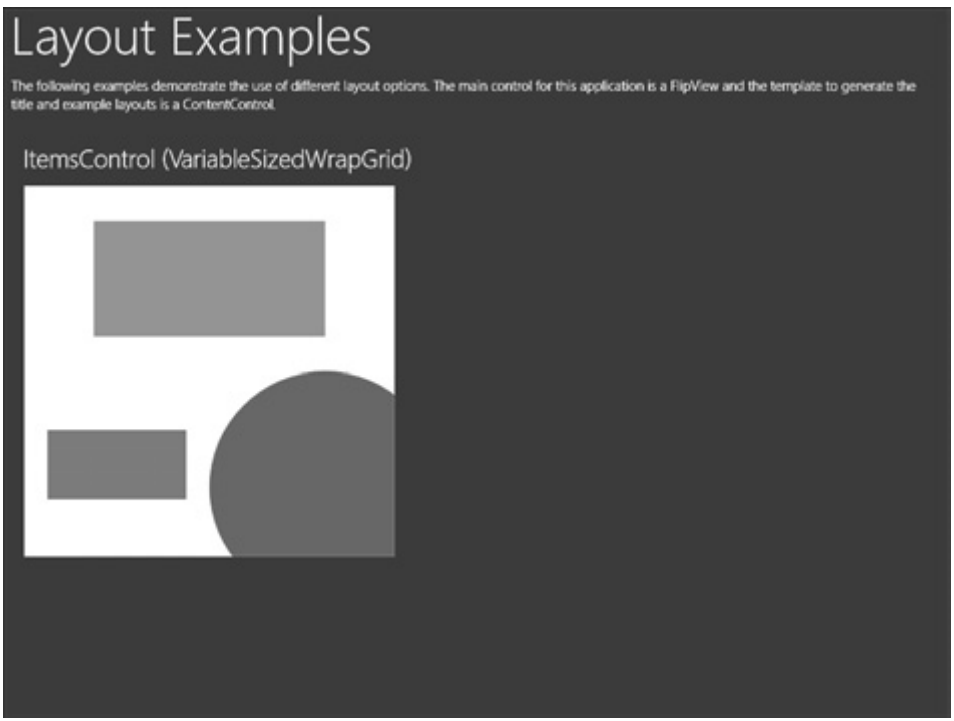


Рис. 3.8. Макет на основе элемента управления VariableSizedWrapGrid

Элемент управления ContentControl

Элемент управления `ContentControl` — это обычный контейнер. Он содержит единственный дочерний элемент и сам по себе для описания макета не используется. Элемент `ContentControl` часто применяют, чтобы обозначить место, где позже может оказаться один или несколько других элементов управления, а также при создании производных элементов управления, рассчитанных на размещение одного элемента с контентом. Важно отметить, что по умолчанию `ContentControl` подстраивается под дочерний элемент.

Для того чтобы увидеть это в действии, вы можете создать сетку (`Grid`) с двумя столбцами одинаковой ширины. Как показано в листинге 3.7,

задайте стиль, предлагаемый по умолчанию для зеленых прямоугольников с синей обводкой, которые могут растягиваться, занимая свободное место.

Листинг 3.7. Определение элемента управления Grid для размещения элементов ContentControl

```
<Grid.Resources>
  <Style TargetType="Rectangle">
    <Setter Property="Fill" Value="Green"/>
    <Setter Property="HorizontalAlignment" Value="Stretch"/>
    <Setter Property="VerticalAlignment" Value="Stretch"/>
    <Setter Property="Stroke" Value="Blue"/>
    <Setter Property="StrokeThickness" Value="3"/>
    <Setter Property="Margin" Value="2"/>
  </Style>
</Grid.Resources>
```

Теперь добавьте два элемента управления `ContentControl`. Оба элемента растягиваются, занимая доступное пространство. Добавьте ко второму элементу специальные присоединенные свойства `HorizontalContentAlignment` и `VerticalContentAlignment`, определяющие выравнивание. Оба эти свойства установите в значение `"Stretch"`. В листинге 3.8 показан код, который должен стать результатом этих действий.

Листинг 3.8. Экземпляры элементов управления ContentControl внутри сетки

```
<ContentControl HorizontalAlignment="Stretch"
                VerticalAlignment="Stretch">
  <ContentControl.Content>
    <Rectangle/>
  </ContentControl.Content>
</ContentControl>
<ContentControl Grid.Column="1"
                HorizontalAlignment="Stretch"
                VerticalAlignment="Stretch"
                HorizontalContentAlignment="Stretch"
                VerticalContentAlignment="Stretch">
  <ContentControl.Content>
    <Rectangle/>
  </ContentControl.Content>
</ContentControl>
```

Выполнив этот код, вы увидите, что выведен лишь самый правый элемент. Это результат особенностей реализации `ContentControl`. Несмотря на то, что контейнер растянулся, чтобы занять все доступное свободное

пространство, он не предоставляет это пространство для дочерних элементов. Если у дочерних элементов не заданы фиксированные высота и ширина, им не будет предоставлено место для автоматического расширения. В итоге, использование свойства `Stretch` (растяжение) в нашем примере приводит к тому, что элемент имеет нулевой размер. Если же заданы параметры выравнивания контента, а именно `HorizontalContentAlignment` и (или) `VerticalContentAlignment`, дочерний элемент позиционируется относительно родительского контейнера. Если в данном случае используется свойство `Stretch`, дочернему элементу будет выделено все свободное пространство, которое он сможет занять.

Элемент управления ItemsControl

Элемент управления `ItemsControl` — это специальный контейнер, предназначенный для размещения коллекций дочерних элементов. Чаще всего он используется с элементами управления, которым нужно выводить на экран списки данных, например обычные и комбинированные списки. В дополнение к панели, которая задает особенности расположения контента, этот элемент управления имеет свойство `ItemSource`. Данному свойству может быть присвоен любой перечисляемый список (реализующий интерфейс `IEnumerable`), предназначенный для контента элемента управления.

Элемент управления ScrollViewer

Элемент управления `ScrollViewer` (средство просмотра с поддержкой прокрутки) включает в себя список других UI-элементов, который можно прокручивать. Элемент `ScrollViewer` содержит виртуальную поверхность, размер которой зависит лишь от количества выводимых элементов. Он вмещает лишь один элемент, который может быть чем угодно, от одиночного элемента управления до панели с дочерними элементами. Если дочерний элемент элемента `ScrollViewer` — это список, содержащий 1000 элементов, каждый из которых имеет высоту 20 единиц, виртуальная поверхность элемента составит 20 000 единиц. «Окно просмотра» — это видимая часть контента элемента `ScrollViewer`, содержащая подмножество полного набора данных, которые видны при прокрутке. Элементы управления `ScrollViewer` могут прокручиваться в вертикальном направлении, в горизонтальном или в обоих.

Главное преимущество элемента `ScrollViewer` заключается в том, что он имеет встроенную поддержку масштабирования и сенсорного управления контентом, причем сенсорное управление поддерживается автоматически.

В том случае, если поддерживается прокрутка контента в нескольких направлениях, элемент может реализовывать концепцию «направляющих», или зон, где прокрутка ограничена каким-то определенным направлением. Это предотвращает смещение изображения, когда пользователь, прокручивая контент, например, по вертикали, слегка сдвигает его и по горизонтали. Видимая область элемента обеспечивает масштабирование контента. Все это можно задать с помощью свойств элемента управления.

Вот пример XAML-кода, который описывает `ScrollViewer`:

```
<ScrollViewer Width="400" Height="400" Background="White">
  <StackPanel Background="White" Width="400" Height="500">
    <Rectangle Fill="Red" Width="250" Height="125"/>
    <Rectangle Fill="Green" Width="150" Height="75"/>
    <Ellipse Fill="Blue" Width="250" Height="250"/>
  </StackPanel>
</ScrollViewer>
```

Результат построения макета, поддерживающего прокрутку, показан на рис. 3.9.

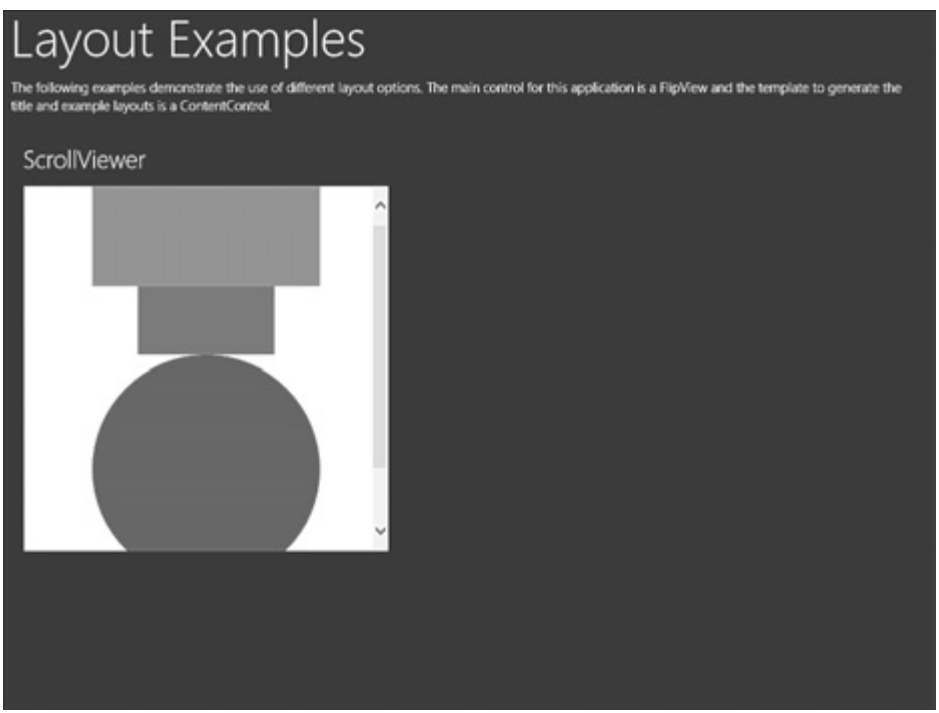


Рис. 3.9. Полоса прокрутки элемента управления StackPanel, размещенного внутри контейнера ScrollViewer

Элемент управления ViewBox

Элемент управления `ViewBox` (окно просмотра) — это уникальный контейнер. Единственное его назначение заключается в изменении размеров контента. Вы можете создать контент, размер которого не задан, а затем точно вписать его в доступное экранное пространство. Элемент управления `ViewBox` позволяет задавать метод управления размером контента. Вот доступные режимы:

- `None`. Размер контента не меняется, при необходимости некоторые части контента отсекаются.
- `Fill`. Соотношение сторон контента может быть изменено, чтобы точно вписаться в контейнер даже за счет искажения.
- `Uniform`. Размер контента меняется, чтобы наилучшим образом вписаться в контейнер. Соотношение сторон контента остается неизменным, поэтому сверху, снизу, справа или слева может появиться пустое пространство.
- `UniformToFill`. Размер контента меняется, чтобы наилучшим образом заполнить контейнер. При этом сохраняется соотношение сторон контента, но если оно не совпадает с соотношением сторон контейнера, часть контента может быть отсечена.

В листинге 3.9 полностью представлена XAML-разметка сетки, которая демонстрирует различные возможности автоматической настройки размера контента. Хотя существует множество других контейнеров и элементов управления, очень важно понять особенности работы именно тех, которые описаны в предыдущих разделах. Они представляют собой базовые классы и шаблоны, а другие элементы либо являются их наследниками, либо их используют.

Листинг 3.9. Разметка на основе элемента управления Grid, демонстрирующая особенности использования контейнера ViewBox

```
<Grid HorizontalAlignment="Right" VerticalAlignment="Top"
    Margin="10" Width="400">
  <Grid.Resources>
    <Style TargetType="TextBlock">
      <Setter Property="Text" Value="Viewbox"/>
      <Setter Property="FontSize" Value="72"/>
      <Setter Property="FontWeight" Value="Bold"/>
    </Style>
```

```
    <Style TargetType="Viewbox">
      <Setter Property="Height" Value="50"/>
      <Setter Property="Margin" Value="5"/>
    </Style>
  </Grid.Resources>
  <Grid.ColumnDefinitions>
    <ColumnDefinition Width="100"/>
    <ColumnDefinition Width="100"/>
    <ColumnDefinition Width="100"/>
    <ColumnDefinition Width="100"/>
  </Grid.ColumnDefinitions>
  <Viewbox Stretch="None">
    <TextBlock/>
  </Viewbox>
  <Viewbox Stretch="Fill" Grid.Column="1">
    <TextBlock/>
  </Viewbox>
  <Viewbox Stretch="Uniform" Grid.Column="2">
    <TextBlock/>
  </Viewbox>
  <Viewbox Stretch="UniformToFill" Grid.Column="3">
    <TextBlock/>
  </Viewbox>
</Grid>
```

Когда вы завершите работу над этой разметкой, то увидите один и тот же результат и в визуальном конструкторе, и при запуске приложения (рис. 3.10).

Рис. 3.10. Различные варианты масштабирования контента в контейнере

Увидеть пример использования элемента `ViewBox` вы можете на последней странице примера Layout. Также вы можете открыть в конструкторе страницу ViewboxExample.xaml из приложения Panels. Когда вы запустите приложение Panels, то увидите множество примеров макетов, предназначенных для управления списками объектов. Первый пример касается элемента управления `GridView`.

Элемент управления GridView

Элемент управления `GridView` (средство просмотра структур на основе сетки) — это мощный элемент управления, который позволяет выводить на экран списки данных в удобном для перемещения формате. Важная особенность этого элемента управления заключается в том, что он поддерживает работу со сгруппированными данными. Это означает, что вы можете группировать большие списки и назначать им подзаголовки для логической организации данных.

В проекте Panels определен простой класс `SimpleItem`, описывающий тип геометрической фигуры. Этот класс включает в себя определения красного, зеленого и синего цветовых компонентов, которые позволяют задавать объектам различные цвета. Второй класс, который называется `SimpleItemsList`, описывает коллекцию объектов `SimpleItem`. В файле ItemDisplay.xaml определен пользовательский элемент управления для вывода данных, который можно рассматривать как многократно используемую коллекцию UI-элементов. В данном случае можно один раз задать внешний вид и поведение элемента, а затем многократно применять его в других местах приложения.

Пользовательский элемент управления благодаря контейнеру `ContentControl` выводит на экран фигуру, отражающую элемент данных. Контейнер с помощью конвертера данных (`Converter`) преобразует объект `SimpleItem` из текущего контекста данных (объекта `DataContext`) в закрашенную геометрическую фигуру. Контейнер в качестве параметра принимает не какое-то отдельное значение, а целый объект `SimpleItem`. Для начала он определяет сплошную кисть, основываясь на заданных в объекте цветовых значениях:

```
var color = Color.FromArgb(0xff, (byte)item.Red,
  (byte)item.Green, (byte)item.Blue);
var brush = new SolidColorBrush(color);
```

Далее он выводит геометрическую фигуру, основываясь на типе элемента. Например, окружность (`Circle`) — это эллипс, высота (`Height`) и ширина (`Width`) которого равны:

```
case ItemType.Circle:
  var circle = new Ellipse();
  circle.Width = 200;
  circle.Height = 200;
  circle.Fill = brush;
  retVal = circle;
  break;
```

На главной странице объявляется специальный элемент, который называется `CollectionViewSource`. Он предназначен для работы с коллекциями, упрощая группирование и выделение элементов. Первая коллекция поддерживает группировку, она объявлена следующим образом:

```
<CollectionViewSource x:Name="CVSGrouped"
  IsSourceGrouped="True"/>
```

Настройка коллекции довольно проста. Источник данных для коллекции — это запрос, который группирует элементы, основываясь на сведениях о представляемых этими элементами геометрических фигурах:

```
CVSGrouped.Source = from item in list
                    group item by item.Type into g
                    orderby g.Key
                    select g;
```

Здесь использован запрос, интегрированный в язык (Language Integrated Query, LINQ). Эта технология подходит для создания запросов к данным любого типа. В данном случае тип данных — простой список. Если вы не знакомы с LINQ, то можете начать изучение этой технологии с книги «Essential LINQ», которую найдете на странице http://www.informit.com/store/product.aspx?isbn=0321564162.

Элемент управления `GridView` привязан к списку в качестве источника данных. Этот элемент управления состоит из нескольких частей, каждая из которых полностью поддается настройке. Например, вы можете настроить правила вывода заголовков групп:

```
<GroupStyle.HeaderTemplate>
  <DataTemplate>
    <TextBlock Text="{Binding Key}"/>
  </DataTemplate>
</GroupStyle.HeaderTemplate>
```

Шаблоны представляют собой многократно используемые фрагменты XAML-разметки. `DataTemplate` — это специальный шаблон, который позволяет настраивать внешний вид и поведение элементов на основе их источника данных. В данном примере заголовок, заданный с помощью `DataTemplate`, содержит простой текстовый элемент, связанный с ключом группы. Приведенный запрос получает этот ключ, текст которого определяет тип фигуры. В итоге в заголовках будут отображаться названия категорий объектов, такие как «Circle» (окружность) или «Rectangle» (прямоугольник).

Еще один шаблон задает особенности группирования и выравнивания отдельных элементов:

```
<GroupStyle.Panel>
  <ItemsPanelTemplate>
    <VariableSizedWrapGrid Orientation="Vertical"/>
  </ItemsPanelTemplate>
</GroupStyle.Panel>
</GroupStyle>
```

С элементом управления `VariableSizedWrapGrid` вы уже знакомы. Как помните, в нем применяется особый способ определения того, нужно ли элементам дополнительное пространство. В данном примере прямоугольники и эллипсы занимают по два столбца, окружности и квадраты — лишь по одному. Новый класс, который называется `ShapeView`, является наследником элемента управления `GridView`:

```
public class ShapeView : GridView
```

Далее осуществляется переопределение метода `PrepareContainerForItemOverride`. Если элемент данных — это экземпляр класса `SimpleItem`, его тип либо `Ellipse`, либо `Rectangle`, то присоединенное свойство `ColumnSpan` устанавливается в значение 2:

```
if (itemdetail.Type.Equals(ItemType.Ellipse) ||
  itemdetail.Type.Equals(ItemType.Rectangle))
{
  element.SetValue(VariableSizedWrapGrid.ColumnSpanProperty, 2.0);
}
```

Так задается расположение элементов в группах. Однако имеется также шаблон, который определяет взаимное положение групп. В этом примере группы расположены горизонтально в элементе управления `VirtualizingStackPanel`:

```
<GridView.ItemsPanel>
  <ItemsPanelTemplate>
    <VirtualizingStackPanel Orientation="Horizontal"/>
  </ItemsPanelTemplate>
</GridView.ItemsPanel>
```

И наконец, отдельные элементы списка отформатированы с помощью шаблона `DataTemplate`, который просто задействует пользовательский элемент управления `ItemDisplay`:

```
<GridView.ItemTemplate>
  <DataTemplate>
    <local:ItemDisplay/>
  </DataTemplate>
</GridView.ItemTemplate>
```

Результат визуализации этого макета приведен на рис. 3.11. Обратите внимание на то, что элементы располагаются в несколько строк и столбцов, а также на то, что группы с эллипсами имеют большую ширину, нежели группы с окружностями. Кроме того, вы можете видеть заголовки групп маленького размера, расположенные в верхней части страницы, а также выделенный элемент. То и другое реализовано встроенными возможностями элемента управления `GridView`.

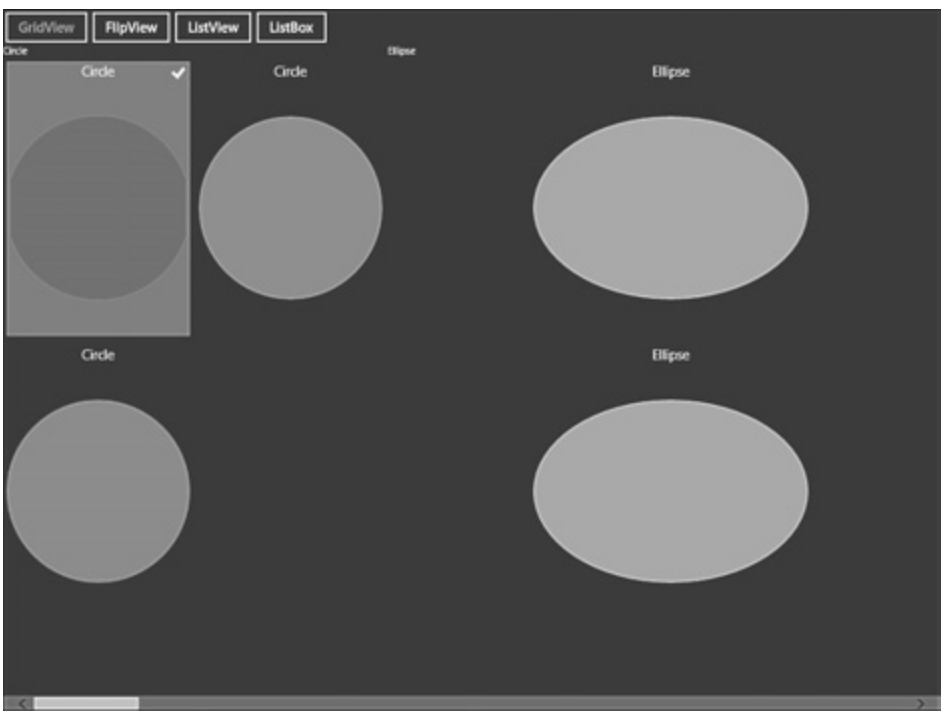


Рис. 3.11. Макет на основе элемента управления GridView

Элемент управления ListView

Элемент управления `ListView` (средство просмотра списков) напоминает `GridView`. Основное различие заключается в том, что его основная

ориентация горизонтальная, а не вертикальная. Он разработан для более узких областей просмотра, например для страниц приложений, рассчитанных на работу в режиме фиксации.

СОВЕТ

Вы узнаете о некоторых нетривиальных возможностях построения макетов в главе 4. Один из примеров — способность переключаться между полноразмерным окном в полноэкранном режиме просмотра и уменьшенным окном в состоянии фиксации. Окно в состоянии фиксации занимает узкую полосу экрана. Обычно переход страницы в этот особый режим просмотра происходит при переключении от элемента GridView на элемент ListView. Подобная трансформация может также происходить при смене ориентации дисплея с портретной на альбомную. Способность приложений адаптироваться к ориентации и разрешению экрана целевого устройства — это фундаментальная особенность приложений для Windows 8.

На рис. 3.12 вы можете видеть те же фигуры, организованные с помощью элемента управления `ListView`.

Рис. 3.12. Макет на основе элемента управления ListBox

Элемент управления FlipView

Элемент `FlipView` — это новый уникальный элемент управления, который поддерживает перелистывание элементов списка. Он показывает один пункт списка и позволяет перемещаться по списку, листая его влево или вправо. В него встроена анимация, автоматически сдвигающая текущий элемент за пределы видимой области, одновременно выводя в эту область следующий элемент. Он удобен для организации перемещения по отдельным элементам, когда каждый из них нужно подробно рассмотреть, либо для случаев, когда список нужно быстро листать, чтобы увидеть каждый из элементов.

Элемент управления ListBox

Элемент управления `ListBox` (классический список) включен сюда для сравнения с более новыми элементами управления (см. рис. 3.12). Он может выводить списки элементов, но не обладает более совершенными возможностями элемента управления `ListView`. Так, он не реализует расширенные возможности выделения и не поддерживает группировку данных. Вероятно, осуществляя перевод существующих приложений на платформу Windows 8, вы решите заменить все имеющиеся в нем элементы управления `ListBox` более функциональными `ListView`.

Обычные элементы управления

Понимая возможности XAML в плане создания макетов, вы можете приступить к разработке пользовательского интерфейса, применяя лишь встроенные элементы управления. Все элементы управления, которые поддерживает Windows 8, находятся в пространстве имен `Windows.UI.Xaml.Controls`. Хотя вы можете просто вставлять теги элементов управления в XAML-разметку, как это делается для других объектов, важно отметить, что большинство элементов управления — это аппаратно-зависимые WinRT-объекты, которые проецируются в XAML-разметку. Это обеспечивает как высокий уровень производительности WinRT, так и гибкость подсистемы XAML.

В табл. 3.4 приведен алфавитный список обычных элементов управления с краткими описаниями. Многие них мы подробно рассмотрим в следующих главах этой книги.

Таблица 3.4. Обычные элементы управления (значком * отмечены новые элементы управления среды WinRT)

Элемент управления	Описание
AppBar* (панель приложения)	Представляет собой панель инструментов для вывода команд приложения. Больше о ней вы узнаете в главе 4
Button (кнопка)	Элемент управления, на котором пользователь может щелкнуть мышью или которого может коснуться. В ответ возникает событие. Также он может быть связан с командой. Больше о командах вы узнаете в главе 8
CheckBox (флажок)	Простой элемент управления, который может быть либо установлен, либо сброшен. Обычно используется для управления логическими (Boolean) значениями
ComboBox (комбинированный список)	Список, в котором пользователь может выбирать позиции. В Windows 8 вы, скорее всего, будете использовать списки, поддерживающие выделение, или сетки с соответствующими возможностями, чтобы полнее контролировать макет элемента и особенности сенсорного взаимодействия с ним
HyperlinkButton (кнопка-гиперссылка)	Выводит соответствующим образом отформатированный текст. Обычно это указывает на то, что на данном тексте можно щелкнуть, чтобы перейти на какую-либо страницу приложения или на веб-ресурс
Image (изображение)	Контейнер для растровых изображений
MediaElement (мультимедиа-элемент)	Рабочая поверхность для воспроизведения видеоклипов и аудиозаписей
MediaPlayer (мультимедиа-проигрыватель)	Элемент управления, обладающий расширенными возможностями, который позволяет воспроизводить мультимедийный контент и управлять этим процессом
PasswordBox (поле ввода пароля)	Специальное поле для ввода текста, которое маскирует введенные символы, чтобы пользователь мог безопасно ввести пароль или какой-либо код
PopupMenu* (всплывающее меню)	Нестандартное меню, содержащее заданные вами команды
ProgressBar (индикатор выполнения)	Индикатор, который позволяет судить о ходе выполнения некоторой операции

Элемент управления	Описание
ProgressRing* (кольцевой индикатор выполнения)	Индикатор выполнения, имеющий вид анимированного кольца
RadioButton (переключатель)	Позволяет выбирать взаимоисключающие элементы из некоторого набора
RepeatButton (кнопка с автоматическим повтором)	Особый вид кнопки, которая непрерывно, пока она нажата, вызывает событие Click. Это отличает ее от обычной кнопки, которая вызывает лишь одно такое событие
RichTextBlock (форматированная текстовая область)	Рабочая поверхность, на которую можно выводить форматированный текст и изображения
ScrollBar (полоса прокрутки)	Полосы прокрутки с ползунками
SemanticZoom* (семантическое масштабирование)	Позволяет пользователю переключаться между двумя представлениями коллекции
TextBlock (текстовая область)	Область вывода обычного текста
TextBox (текстовое поле)	Поле, которое дает пользователю возможность вводить однострочный или многострочный текст
Slider (ползунок)	Элемент управления в виде шкалы с ползунком, которая позволяет пользователю задавать значения, перемещая ползунок
ToggleButton (кнопка-выключатель)	Кнопка, которая может переключаться между двумя состояниями
ToggleSwitch выключатель)	Выключатель, который может пребывать в одном из двух состояний
ToolTip (всплывающая подсказка)	Контекстно-зависимое окно, которое появляется для вывода дополнительных сведений об элементе
WebView* (средство просмотра HTML-контента)	Контейнер для веб-контента

Все эти элементы управления можно размещать на XAML-страницах путем перетаскивания, создавая с их помощью пользовательский интерфейс

приложения. Также можно создавать нестандартные элементы управления или группировать существующие, многократно использовать их, применяя собственные элементы управления и шаблоны. Этот раздел содержит лишь краткое описание элементов управления, в следующих главах мы рассмотрим некоторые из них гораздо подробнее.

Выводы

Эта глава посвящена XAML, специальному языку разметки, который служит для описания пользовательских интерфейсов приложений для Windows 8. Вы узнали о зависимых и присоединенных свойствах, о том, как они дополняют наборы базовых CLR-свойств, расширяя возможности XAML. Затем вы узнали, как привязка данных позволяет отделить пользовательский интерфейс и логику уровня представления от нижележащих данных приложения.

Раскадровки — это инструмент анимации элементов на экране. Стили и ресурсы позволяют создать централизованное хранилище для данных, описывающих внешний вид и особенности поведения приложения, или их «темы». И наконец, элементы управления позволяют по-разному организовать элементы и наборы элементов на странице.

В следующей главе вы узнаете об уникальных особенностях приложений для Windows 8. Вы воспользуетесь знаниями о макетах и элементах управления, чтобы создать приложение, которое реагирует на смену ориентации или разрешения экрана. Это можно сделать с помощью специального класса, который называется диспетчером визуальных состояний. Вы узнаете, как в приложениях для Windows 8 обрабатывать ввод данных пользователем, как подключить панель приложения, как работать с контекстными меню, как создать специальную страницу «О программе». И наконец, вы создадите пригодные для многократного использования WinRT-компоненты, которые можно задействовать в качестве строительных блоков при построении более сложных приложений.

Приложения для Windows 8

4

Все приложения для Windows 8 обладают общими свойствами, которые позволяют им отлично работать на разных устройствах, на которых установлена эта операционная система. Это быстрые и гибкие приложения, которые легко адаптируются к различным режимам просмотра и экранам различных размеров. С ними можно взаимодействовать, используя различные способы ввода данных. Они поддерживают сенсорное взаимодействие с пользователем и предоставляют ему единообразный интерфейс, в котором для управления приложением предусмотрены стандартные места размещения команд и контекстных меню.

В предыдущей главе вы познакомились с XAML, декларативным языком описания разметки, который позволяет отделить дизайн от программирования, что дает возможность создавать мощные и расширяемые пользовательские интерфейсы. В этой главе вы узнаете, как посредством XAML-разметки и исполняемого программного кода наделить приложения для Windows 8 разными уникальными возможностями. Причем многие из этих возможностей доступны для использования и настройки непосредственно из XAML-разметки.

Макеты и режимы просмотра

Приложения для Windows 8 могут работать с различными макетами и режимами просмотра. В частности, доступны различные ориентации устройства (портретная и альбомная), а также состояния фиксации и заполнения,

когда окна двух приложений делят экран, располагаясь рядом друг с другом. Встроенные шаблоны Visual Studio 2012 дают приложениям возможность переключаться между различными режимами просмотра автоматически. Воспользовавшись встроенным имитатором, можно без проблем протестировать механизм смены макета приложением, даже если у вас нет планшетного компьютера или акселерометра.

Имитатор

Откройте пример Windows8Application к главе 4 этой книги. Если вы еще не загрузили примеры, их можно найти на странице http://windows8applications.codeplex.com/.

Вместо того чтобы начинать отладку приложения на локальном компьютере, как это происходит по умолчанию, выберите в раскрывающемся списке режимов отладки вариант Simulator (имитатор), как показано на рис. 4.1.

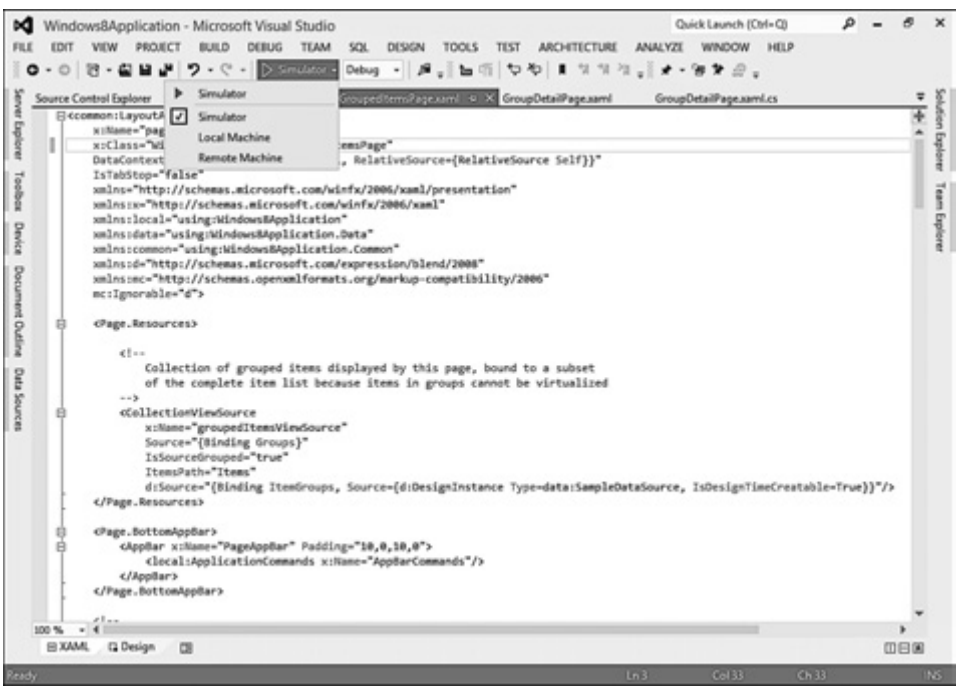


Рис. 4.1. Подготовка к отладке приложения в имитаторе

Имитатор упрощает тестирование различных вариантов использования приложения, таких как выполнение на экранах с разным разрешением и в различных режимах просмотра. Кроме того, он предоставляет имита-

цию сенсорного экрана в том случае, если вы занимаетесь разработкой приложения на компьютере, дисплей которого не является сенсорным. Окно имитатора оформлено в стиле, делающем его похожим на планшетный компьютер с клавишей Windows. С правой стороны окна имитатора расположен набор кнопок для доступа к различным функциям, в том числе (сверху вниз):

1. Mouse Mode (режим мыши). Позволяет использовать указатель мыши обычным способом.
2. Touch Emulation (базовый сенсорный режим). Имитация сенсорного взаимодействия с устройством с помощью мыши. Для того чтобы выполнить жест касания, достаточно щелкнуть в нужном месте левой кнопкой мыши.
3. Touch Emulation Pinch and Zoom (сенсорный режим с имитацией жестов масштабирования). В этом режиме можно протестировать жесты масштабирования (сжатия и расширения). Для их выполнения достаточно нажать левую кнопку мыши и, не отпуская ее, вращать колесо мыши для увеличения или уменьшения масштаба. То же самое можно выполнить и в режиме мыши (Mouse Mode), удерживая нажатой клавишу Ctrl и вращая колесо мыши.
4. Touch Emulation Rotate (сенсорный режим с имитацией жеста вращения). Режим имитации вращения реализуется удержанием левой кнопки мыши и вращением ее колеса.

Rotate 90 Degrees Clockwise (вращение по часовой стрелке на 90 градусов). Эта команда позволяет повернуть окно имитатора на 90° по часовой стрелке. В результате меняется ориентация экрана имитируемого устройства.

Rotate 90 Degrees Counterclockwise (вращение против часовой стрелки на 90 градусов). Данная команда осуществляет поворот окна имитатора на 90 градусов против часовой стрелки. Аналогично предыдущей команде, это приводит к смене ориентации экрана имитируемого устройства.

1. Change Resolution (смена разрешения). Позволяет менять разрешение экрана имитируемого устройства. С помощью этой команды можно установить разрешение экрана виртуального устройства даже большим, чем физическое разрешение дисплея, на котором вы работаете.
2. Set Location (задание положения). С помощью этой команды можно задать географическое местоположения имитируемого устройства, установив широту и долготу, высоту над уровнем моря и погрешность измерений.
3. Screenshot (получение снимка экрана). Эта удобная команда позволяет делать снимки экрана имитатора в его текущем разрешении.

4. Configure Screenshot (настройка снимка экрана). С помощью данной команды можно указать, нужно ли помещать копии экрана имитатора в буфер обмена или сохранять в виде файлов по заданному пути.

5. Help (справка). Вывод справочной информации об имитаторе.

Имитатор обладает богатыми возможностями. Установите разрешение окна имитатора в 1024×768. Вы увидите обычное приложение, содержащее элементы и группы элементов, как показано на рис. 4.2. Действие имитатора основано на подключении удаленного рабочего стола к устройству, на котором вы исполняете приложение (что отличается от принципов работы виртуальной машины). Это позволяет вам в полной мере протестировать приложение на собственном компьютере, применяя различные варианты сенсорного взаимодействия с приложением и различные параметры дисплея.

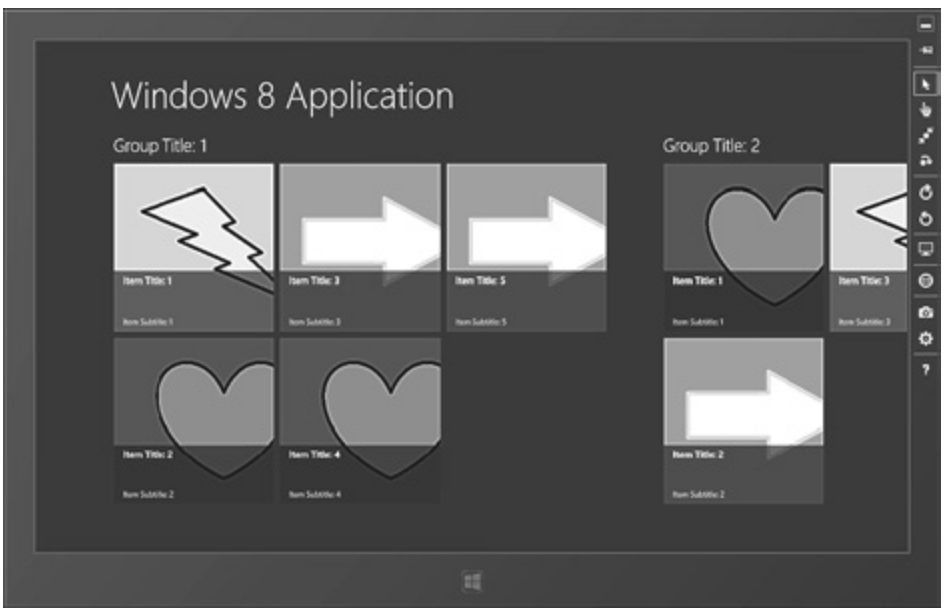


Рис. 4.2. Приложение запущено в имитаторе

Убедившись в том, что приложение имеет фокус ввода (вы можете щелкнуть мышью в свободном пространстве окна приложения или коснуться его, чтобы гарантировать, что это так), удерживая клавишу Windows, нажмите клавишу . (точка). Это заставит приложение перейти в режим фиксации. Если при этом исполняется еще какое-нибудь приложение для Windows 8, его оно займет оставшееся (большее по размеру) пространство экрана. В противном случае эта часть экрана останется пустой, но и в этом случае

окно текущего приложения займет узкую область экрана, как показано на рис. 4.3.

Рис. 4.3. Приложение в режиме фиксации

Режим фиксации — это особый режим, предназначенный для того, чтобы пользователь мог быстро просмотреть данные какого-либо приложения во время работы другого приложения. Для этого режима лучше всего подходит вертикально ориентированный элемент управления `ListView`, а не горизонтальный `GridView`. Приложения могут работать подобным образом лишь тогда, когда ширина экрана составляет как минимум 1366 пикселов. При меньшем разрешении экрана, например 1024×768, этот режим недоступен. Режим фиксации — это полноценный режим просмотра приложения, хотя область просмотра ограничена значением 320 пикселов в ширину. Этого вполне достаточно для решения большого числа задач, а для смартфонов подобная ширина экрана вполне обычна.

Помимо места, отведенного для окна приложения, в режиме фиксации на экране имеется разделитель размером 22 пиксела. Пользователь может с его помощью перетаскивать окно приложения по экрану. В итоге свободной может остаться область шириной 1024 пиксела, и приложение, работавшее в режиме заполнения, получает в свое распоряжение область размером 1024×768 пикселов (1366 – 320 – 22 = 1024). В случае приложений для Windows 8 это минимально рекомендованное разрешение.

Как вы можете заметить, внешний вид приложения при переходе в режим фиксации меняется. Вместо горизонтально ориентированной сетки, которая содержит элементы, распределенные по строкам и столбцам, на

странице остается вертикальный список. В узкой области просмотра это упрощает навигацию по списку и выбор элементов. Однако гораздо интереснее то, как программа реализует подобное изменение. XAML-разметка и программный код, встроенные в шаблон Grid Application (табличное приложение), используют для этого мощный класс, который называется диспетчером визуальных состояний.

Диспетчер визуальных состояний

Диспетчер визуальных состояний (Visual State Manager, VSM) позволяет отделить управление внешним видом и особенностями поведения пользовательского интерфейса приложения для Windows 8 от его программной логики. Он работает совместно с системой привязки данных, отделяя все, что связано с пользовательским интерфейсом, от остальной логики приложения.

Диспетчер визуальных состояний обрабатывает логику состояний и переходов элементов управления. Понятие «элемент управления» в данном случае не ограничено нестандартными элементами управления или шаблонами элементов управления. Диспетчер одинаково хорошо подходит для управления состоянием страниц и пользовательских элементов управления. VSM работает только с элементами, которые являются наследниками класса `Control`. Среди них страницы (элементы класса `Page`) и пользовательские элементы управления (`UserControl`). Диспетчер визуальных состояний всегда должен представлять собой корневой элемент шаблона элемента управления. В проекте Windows8Application вы обнаружите элемент `VisualStateManager.VisualStateGroups`, который вложен в главную сетку приложения в файле GroupedItemsPage.xaml.

VSM позволяет описывать группы, состояния и, при необходимости, переходы. Группа — это набор взаимоисключающих состояний. Как можно судить по названию, группы позволяют создавать связи между родственными состояниями. В примере, который мы рассматриваем, страница приложения может пребывать в одном из обычных состояний, относящихся к группе `ApplicationViewStates`. Вот эти состояния:

- `FullScreenLandscape` (полноэкранный альбомный режим). Приложение исполняется в полноэкранном режиме, планшетный компьютер находится в альбомной ориентации.
- `Filled` (режим заполнения). Приложение занимает большую часть экрана, в то время как меньшая его часть занята приложением, которое исполняется в режиме фиксации.

- `FullScreenPortrait` (полноэкранный портретный режим). Приложение исполняется в полноэкранном режиме, планшетный компьютер находится в портретной ориентации.
- `Snapped` (режим фиксации). Приложению, которое исполняется в режиме фиксации, выделяется область с краю экрана шириной в 320 пикселов.

Важно понимать, что эти состояния являются взаимоисключающими, так как VSM позволяет странице в некоторый момент времени пребывать лишь в одном из них. Группы отражают контракт, касающийся особенностей режимов работы страницы. Вы задаете набор состояний, в которых может пребывать страница, и поддерживаете функциональность страницы в этих состояниях. Управление состояниями осуществляется с помощью базового класса, от которого унаследована страница. VSM позволяет настраивать визуальные эффекты, происходящие при переходе страницы в то или иное состояние.

Для того чтобы увидеть, как осуществляется управление состояниями, откройте папку `Common` и взгляните на класс `LayoutAwarePage`. Этот класс представляет собой часть шаблона проекта, он применяется при обработке события `ViewStateChanged`, которое вызывается всякий раз, когда меняется ориентация экрана устройства либо происходит переключение между режимами просмотра в пределах неизменной ориентации экрана. Обработчик события получает строковое представление нового состояния просмотра и указывает VSM на то, что нужно перейти в это состояние:

```
VisualStateManager.GoToState(layoutAwareControl, visualState, false);
```

Обратите внимание, в смене состояния просмотра никак не задействована логика пользовательского интерфейса. Это и есть отделение логики приложения от логики пользовательского интерфейса. Приложение занимается управлением состояниями, а изменения в пользовательском интерфейсе, которые соответствуют различным состояниям, описаны в XAML-коде с помощью раскадровок. Если вы задействуете несколько логических групп состояний в применении к одному элементу, это означает, что для данного элемента одновременно работают несколько раскадровок.

Важное правило для групп заключается в том, что они не должны зависеть друг от друга. Хотя элемент управления может существовать одновременно в нескольких состояниях (точнее, к нему может быть применено одно состояние из каждой группы), визуальные эффекты, заданные в разных группах, не должны накладываться друг на друга. Системные средства не обеспечивают выполнение этого правила, так как в каждом из состояний

вы можете описать любые раскадровки. Однако если не соблюдать это правило, результаты могут оказаться совершенно непредсказуемыми.

Группы — это контейнеры для связанных взаимоисключающих состояний. А что такое *состояние*? У этого понятия есть как логическое, так и физическое толкование. Логически оно описывает взаимоисключающий статус элемента. Физически состояние отражает набор визуальных элементов и параметров. Очень важна возможность отделить детали того, как внешне выглядит элемент, пребывающий в некотором состоянии, от самого этого состояния. В программном коде вы можете просто установить элемент управления в некоторое состояние и позволить VSM выполнить перевод элемента в это состояние. Это упрощает не только создание и тестирование элементов управления, но и их расширение и настройку.

Взгляните на страницу GroupedItemsPage.xaml. Здесь объявлена одна безымянная группа, а первое объявленное в ней состояние выглядит так:

```
<VisualState x:Name="FullScreenLandscape"/>
```

Описание состояния не содержит других сведений. Это значит, что существующего XAML-описания страницы достаточно, чтобы визуализировать пользовательский интерфейс для данного состояния. Если вы взглянете на XAML-код, то увидите элемент управления `SemanticZoom`, который объявлен в первой строке сетки (больше о семантическом масштабировании вы узнаете далее в этой главе). В той же строке есть элемент управления `ScrollViewer`, но его свойство `Visibility` установлено в значение `Collapsed`, то есть он, по умолчанию, не визуализируется. Он содержит элемент управления `ListView`, который применяется в режиме фиксации.

Взгляните теперь на описание состояния `Snapped`. Оно приведено в листинге 4.1. Здесь имеется один элемент управления `Storyboard`, содержащий несколько анимаций. Анимации для перемещения элемента не используются. Они служат для модификации значений зависимых свойств. Диспетчер визуальных состояний с помощью анимаций задает значения свойств элементов управления (скоро мы узнаем, почему).

Листинг 4.1. Описание состояния фиксации

```
<VisualState x:Name="Snapped">
                <Storyboard>
                   <ObjectAnimationUsingKeyFrames
  Storyboard.TargetName="backButton"
  Storyboard.TargetProperty="Style">
                    <DiscreteObjectKeyFrame KeyTime="0"
```

```
  Value="{StaticResource SnappedBackButtonStyle}"/>
                </ObjectAnimationUsingKeyFrames>
                <ObjectAnimationUsingKeyFrames
  Storyboard.TargetName="pageTitle"
  Storyboard.TargetProperty="Style">
                    <DiscreteObjectKeyFrame KeyTime="0"
  Value="{StaticResource SnappedPageHeaderTextStyle}"/>
                </ObjectAnimationUsingKeyFrames>

                <ObjectAnimationUsingKeyFrames
  Storyboard.TargetName="itemListScrollViewer"
  Storyboard.TargetProperty="Visibility">
                    <DiscreteObjectKeyFrame KeyTime="0"
  Value="Visible"/>
                </ObjectAnimationUsingKeyFrames>
                <ObjectAnimationUsingKeyFrames
  Storyboard.TargetName="semanticViewer"
  Storyboard.TargetProperty="Visibility">
                    <DiscreteObjectKeyFrame KeyTime="0"
  Value="Collapsed"/>
                </ObjectAnimationUsingKeyFrames>
              </Storyboard>
</VisualState>
```

В этом примере меняются стили кнопки Back (назад) и заголовка. Для соответствующего свойства элемента управления `SemanticZoom` записывается значение `Collapsed`, то есть этот элемент управления и его дочерние элементы в этом режиме на экран не выводятся. У элемента управления `ScrollViewer` свойство `Visibility` установлено в значение `Visible`, в итоге он вместе с содержащимися в нем элементами в данном режиме выводится на экран. Если свойство `Visibility` элемента управления установлено в значение `Collapsed`, он не выводится в визуальном дереве элементов, если в значение `Visible` — выводится, причем даже в том случае, если его свойство непрозрачности равно 0 или он имеет прозрачный (`Transparent`) цвет. Анимация для конкретного визуального состояния будет исполняться до тех пор, пока страница выводится в этом состоянии. Когда состояние меняется, раскадровка (`Storyboard`) останавливается, значения свойств сбрасываются в состояния, предлагаемые по умолчанию (или принимают новые значения, определяемые раскадровкой для нового состояния).

Диспетчер визуальных состояний использует элемент управления `Storyboard`, так как он имеет наивысший приоритет при изменении значе-

ний зависимых свойств. Понять особенности управления состояниями — значит, понять особенности работы с элементами управления `Storyboard`. Когда некий элемент управления переходит в конкретное состояние, VSM останавливает исполнение действий, заданных элементами управления `Storyboard` для других состояний в той же самой группе (не забудьте, состояния в группе являются взаимоисключающими), а затем начинает исполнять действия, заданные элементом `Storyboard` для нового целевого состояния.

Обычно исходное состояние элемента указывают при его инициализации, чтобы элемент управления присутствовал в «графе состояний» или в коллекции правильных состояний. Класс `LayoutAwarePage` делает это в методе `StartLayoutUpdates`. Обратите внимание на вызов `GoToState`, который выполняется перед выходом из метода.

Переходы делают диспетчер визуальных состояний гибким, позволяя ему управлять тем, как элементы переходят от состояния к состоянию. Вы можете задать переход, применяемый всякий раз, когда элемент управления переходит к определенному состоянию, или ограничить его применение только ситуацией, когда элемент переходит из некоторого заданного состояния в любое другое. Переходы — это очень мощный механизм.

В случае самого простого перехода значения существующей раскадровки служат для анимации смены состояний на основе времени перехода. Вы задаете это время, а диспетчер визуальных состояний делает все остальное. Кроме того, можно создавать собственные раскадровки для переходов. Диспетчер визуальных состояний остановит любую раскадровку перехода, как только элемент управления перейдет в новое состояние. Но он не остановит раскадровку во время изменения состояния.

Семантическое масштабирование

Элементы, видимые на главном экране приложения, которое мы рассматриваем в качестве примера, организованы в группы. Вы можете заметить несколько списков, которые содержат множество элементов, формирующих десятки групп. Хотя встроенные элементы управления разработаны для поддержки больших объемов данных, пользователю может быть не совсем удобно перемещаться по подобному списку. Для решения проблемы можно прибегнуть к *семантическому масштабированию* (semantic zoom).

Семантическое масштабирование (его еще называют контекстным масштабированием) — это методика, которая дает пользователю возможность

уменьшить масштаб представления списка и осуществлять навигацию по нему на более общем уровне средствами предоставленного вами интерфейса. Это не визуальное масштабирование, так как когда пользователь меняет масштаб вывода списка, фактически меняется реальный интерфейс. Если в примере Windows8Application уменьшить масштаб списка в окне приложения, либо коснувшись двумя пальцами сенсорного экрана и разведя их, либо вращая колесо мыши, удерживая клавишу Ctrl, либо воспользовавшись клавишей –, вы увидите, как макет с сеткой, представляющий группы и содержащиеся в них элементы, превратится в макет, состоящий из плиток с названиями групп.

Новое представление данных применяется, когда вы уменьшаете масштаб вывода на экран. Это режим общего представления сменяется прежним режимом детального просмотра, когда вы увеличиваете масштаб. Окно приложения со списком в режиме общего просмотра показано на рис. 4.4. В этом режиме легко видеть весь список групп. Если вы прикоснетесь к плитке, представляющей любую группу, автоматически произойдет переход в режим детального просмотра, а фокус окажется на элементах выбранной группы.

Рис. 4.4. Режим общего представления

Этот мощный механизм можно задействовать различными способами. Приложение для чтения лент новостей может в режиме общего представления показывать названия лент, давая пользователю возможность просматривать новости конкретной ленты в режиме детального просмотра. Приложение для работы со списками контактных сведений может выводить на экран первые буквы фамилий, позволяя быстро перейти к разделу с данными о нужных респондентах. С концептуальной точки зрения вы просто показываете пользователю два разных варианта представления данных, основываясь на том уровне детализации, переход к которому он инициировал, прибегнув к семантическому масштабированию.

Семантическое масштабирование реализуют с помощью элемента управления `SemanticZoom`. Откройте файл GroupedItemsPage.xaml в примере Windows8Application. Найдите в разметке описание элемента управления `SemanticZoom`. Режим `SemanticZoom.ZoomedInView` представляет собой обычный режим просмотра контента, который применяется, когда пользователь работает с приложением. В данном примере я просто взял шаблон, предлагаемый по умолчанию, добавил элемент управления `SemanticZoom` и переместил элемент управления `GridView` в раздел `ZoomedView`. Когда пользователь выполняет жест уменьшения масштаба, происходит переход на режим просмотра `SemanticZoom.ZoomedOutView`. Для данного режима я создал набор плиток, основываясь на кратком руководстве Microsoft, которое доступно по адресу http://msdn.microsoft.com/ru-ru/library/windows/apps/xaml/hh781234.aspx.

Вы также можете загрузить пример, относящийся к элементу управления `SemanticZoom` из Windows SDK, на странице http://code.msdn.microsoft.com/windowsapps/GroupedGridView-77c59e8e.

Работая над нашим примером, я изменил код разметки, чтобы на экран выводились названия групп. Вы можете видеть код объявления плиток в файле словаря MyStyles.xaml. Также я использовал событие `OnNavigatedTo`, чтобы подключить источник данных для `GridView`. Взгляните на файл вспомогательного кода, который называется GroupedItemsPage.xaml.cs, и вы найдете там следующий метод:

```
protected override void OnNavigatedTo(NavigationEventArgs e)
{
  this.DefaultViewModel["Groups"] = e.Parameter;
  this.groupGridView.ItemsSource =
    this.groupedItemsViewSource.View.CollectionGroups;
  base.OnNavigatedTo(e);
}
```

Этот код получает набор групп из источника `CollectionViewSource` в XAML и назначает их источником данных для элемента управления `GridView` в `ZoomedOutView`. Это все, что нужно для использования данного элемента управления. Он за счет собственных встроенных механизмов реагирует на сенсорные жесты масштабирования (управлять масштабированием также можно, нажав клавишу Ctrl и либо прокручивая колесо мыши, либо нажимая клавиши + и -), переключаясь между режимами просмотра контента. Когда пользователь касается какого-либо элемента в режиме общего просмотра (`ZoomedOutView`), элемент управления автоматически переключается в режим детализированного просмотра (`ZoomedInView`) и показывает выбранную группу.

В Microsoft подготовили следующие рекомендации по использованию элемента управления `SemanticZoom`:

- Задайте подходящие размеры целей касания для интерактивных элементов (о целях касания мы поговорим в следующем разделе).
- Создайте удобную интуитивно понятную область масштабирования.
- Используйте структуру, свойственную для данного режима просмотра, например:
  - применяйте групповые названия для элементов групповой коллекции;
  - применяйте упорядочение элементов для несгруппированных коллекций;
  - применяйте страницы для представления коллекций документов.
- Ограничьте количество страниц или экранов в режиме общего просмотра до трех, чтобы пользователь смог быстро переходить к нужным данным.
- Убедитесь, что направление прокрутки контента в различных режимах масштабирования не меняется.
- Не используйте масштабирование для вывода разных наборов элементов (например, показывая один набор элементов в режиме детального просмотра, а другой — в режиме общего просмотра).
- Не задавайте границу для дочерних элементов управления, устанавливайте ее лишь для самого элемента управления `SemanticZoom`.

Полное руководство вы можете найти на странице http://msdn.microsoft.com/ru-ru/library/windows/apps/hh465319.aspx.

Элементы управления `SemanticZoom` и `GridView` автоматически обрабатывают взаимодействие с пользователем. Однако создавая собственные элементы управления и интерфейсы, вы столкнетесь с необходимостью самостоятельно обрабатывать ввод данных пользователем. Для этого важно понимать, как приложения для Windows 8 обрабатывают различные события ввода данных и какова должна быть ожидаемая реакция на эти события. Об этом мы поговорим в следующем разделе.

Обработка пользовательского ввода

Пользовательский ввод — это важный компонент WinRT-приложений. В Windows 8 в качестве первичного способа взаимодействия с пользователем применяется сенсорный ввод, но система поддерживает также множество других способов ввода данных.

Приложения, спроектированные с учетом особенностей сенсорного взаимодействия, полностью поддерживают его и не содержат элементов, работать с которыми можно лишь с помощью мыши, клавиатуры или других средств ввода. Подход к сенсорному управлению приложениями отличается от традиционного подхода по нескольким причинам.

Поскольку Windows 8 поддерживает мультисенсорный ввод, ваше приложение должно быть способно реагировать на одновременное касание сенсорного экрана в нескольких местах. При разработке приложения нужно учитывать особенности создания целей касания, то есть тех мест в приложении, которые реагируют на сенсорный ввод, так как касания экрана не обеспечивают ту же точность, что и указатель мыши. Допустимый предел погрешности при работе с областью, которой касается пользователь, больше, чем при работе с указателем мыши.

Поддержка сенсорного ввода дает пользователям возможность работать с устройством различными способами. В табл. 4.1 приведен список типичных способов взаимодействия.

Работая со стандартными элементами управления в Windows 8, вы обнаружите, что они обладают встроенной поддержкой стандартных способов сенсорного взаимодействия. Хотя в Windows 8 сенсорный ввод является первичным способом ввода, важно поддерживать и другие способы, которые реализуются с помощью стилуса (пера), мыши, клавиатуры. В табл. 4.2 перечислены некоторые основные действия и способы их выполнения с использованием различных режимов ввода данных.

Таблица 4.1. Способы сенсорного взаимодействия с устройством

Способ	Описание
Tap (касание)	Пользователь быстро касается экрана одним пальцем и затем убирает палец
Hold (нажатие и удержание)	Пользователь касается экрана одним пальцем, оставляя палец в месте касания
Drag (перетаскивание)	Пользователь касается экрана одним или несколькими пальцами, затем перемещает их в одном направлении, не убирая с экрана
Pinch/Zoom (сведение и разведение пальцев)	Пользователь касается экрана, как минимум, двумя пальцами, после чего либо сводит их, либо разводит, не убирая с экрана
Rotate (вращение)	Пользователь касается экрана, как минимум, двумя пальцами, после чего осуществляет вращательное движение, не убирая их с экрана
Cross-slide (скольжение поперек)	Пользователь прикасается к объекту на экране, после чего перетаскивает его в направлении, перпендикулярном направлению сдвига контента. Например, в списке с макетом на основе сетки, который можно прокручивать справа налево или наоборот, жест скольжения поперек выполняется сверху вниз или снизу вверх

Таблица 4.2. Режимы ввода данных

Действие	Сенсорный ввод, перо, стилус	Мышь	Клавиатура
Изменение фокуса	Нет эквивалента	Нет эквивалента	Клавиша Tab, клавиши со стрелками
Вызов контекстного меню	Нажатие и удержание	Щелчок правой кнопкой	Клавиша вызова меню
Перетаскивание объектов	Выделение жестом скольжения поперек и перетаскивание на пороговое расстояние из строки или столбца, в котором расположен элемент	Щелчок левой кнопкой и перетаскивание	Сочетания клавиш Ctrl+C (вырезать) и Ctrl+V (вставить)
Запуск или активация	Касание	Щелчок левой кнопкой	Клавиша Enter

продолжение ⇩

Таблица 4.2 *(продолжение)*

Действие	Сенсорный ввод, перо, стилус	Мышь	Клавиатура
Поворот	Сенсорный жест поворота	Использование экранных элементов управления, таких как кнопки, для вращения объектов	Сочетания клавиш Ctrl+. (точка), Ctrl+, (запятая)
Прокрутка	Скользящее движение	Использование полосы прокрутки или колеса мыши	Клавиши со стрелками
Прокрутка на большое расстояние	Скользящее движение с инерцией	Использование бегунка полосы прокрутки	Клавиши PageUp и PageDown
Выделение	Скольжение по диагонали	Щелчок правой кнопкой	Клавиша пробела
Вызов всплывающей подсказки	Нажатие и удержание	Наведение указателя мыши на объект	Перевод фокуса и ожидание
Масштабирование	Жесты сжатия и расширения	Нажатие клавиши Ctrl и прокрутка колеса мыши	Сочетания клавиш Ctrl++ и Ctrl+–

Windows 8 облегчает работу с событиями, поступающими от разных устройств, предоставляя набор аппаратно-независимых *указующих событий* (pointer events). Указующие события происходят, когда пользователь применяет для взаимодействия с приложением сенсорный дисплей, стилус, мышь или клавиатуру. Для того чтобы увидеть примеры, откройте приложение Touch. Это приложение иллюстрирует различные варианты сенсорного взаимодействия с приложением.

Указующее событие

Экран приложения Touch разделен на две части. Левая часть — это обычный список с описанием различных сенсорных событий. В правой части расположены графические объекты, в том числе фигуры, растровые изображения и текстовый фрагмент. Эта часть макета настроена таким образом, чтобы реагировать на различные события ввода данных. Вы можете уменьшать и увеличивать объекты, вращать их и перетаскивать.

Скомпилируйте, установите и запустите приложение. Возможно, вам понадобится воспользоваться имитатором, чтобы поэкспериментировать с различными способами ввода данных, включая сенсорный. Вы довольно скоро увидите, что независимо от выбранного способа взаимодействия с приложением вызывается всего несколько событий:

- `Tapped` (касание). Вызывается при касании сенсорного экрана пальцем, стилусом или при щелчке мышью.
- `DoubleTapped` (двойное касание). Вызывается при двойном касании экрана или двойном щелчке мыши.
- `PointerPressed` (опускание указателя). Вызывается при касании сенсорного экрана пальцем, стилусом или при щелчке мышью.
- `PointerMoved` (перемещение указателя). Вызывается при перетаскивании указателя, когда палец пользователя или стилус не отрывается от экрана, либо при перетаскивании мыши с нажатой левой кнопкой.
- `PointerReleased` (подъем указателя). Вызывается, когда палец пользователя или стилус отрывается от экрана либо отпускается нажатая левая кнопка мыши.

Эти события охватывают подавляющее большинство вариантов взаимодействия пользователя с приложением, помимо прокрутки и перемещения по нему. Если в вашем приложении эти основные события задействованы, можете быть уверены в том, что оно будет единообразно реагировать на применение пользователем различных устройств ввода. Существуют, однако, некоторые указующие события, которые при сенсорном взаимодействии не имеют смысла. События `PointerEntered` и `PointerExited` вызываются, когда указатель перемещается в область, которая настроена на прослушивание соответствующего события или покидает ее. Эти события могут быть вызваны для стилуса и мыши, так как происходит постоянное отслеживание позиции указателя даже тогда, когда не осуществлен захват указателя (то есть не нажата левая кнопка мыши либо стилус не касается сенсорной поверхности). Среди событий сенсорного взаимодействия у них нет эквивалентов, так как подобные события распознаются только тогда, когда пользователь касается сенсорной поверхности экрана.

С учетом сказанного, приложение должно в первую очередь ориентироваться на сенсорное управление. Если вы разработаете основные функциональные возможности своего приложения в расчете на события, которые вызывает мышь или стилус, пользователь не сможет работать с приложением на устройствах, оснащенных лишь сенсорным дисплеем и не имеющих других устройств ввода. Любой тип взаимодействия, который характерен для какого-либо устройства ввода, должен поддерживаться также другими

устройствами. Вы увидите пример этого, когда узнаете о манипуляциях в следующем разделе.

События манипуляции

Манипуляции (управляющие жесты) — это события сенсорного взаимодействия, которые предусматривают использование одного или нескольких пальцев. Манипуляция начинается при первом соприкосновении пальца с сенсорной поверхностью и заканчивается, когда последний палец отрывается от поверхности. Жест касания — это простая форма манипуляции, которая включает в себя прикосновение, очень короткое удержание и отрыв пальца от сенсорной поверхности. Сдвиг — это более сложный жест, так как включает в себя прикосновение, перетаскивание или перемещение в направлении прокрутки и последующее отпускание пальца. Жесты масштабирования — это еще более сложные манипуляции, которые основаны на изменении расстояния между несколькими точками касания.

При манипуляциях должна учитываться инерция. Для того чтобы сенсорное управление было интуитивно понятным, нужно учитывать скорость перемещения пальцев пользователя. Например, медленный сдвиг — это способ точной, медленной прокрутки списка. Если же пользователь начинает выполнять жест сдвига очень быстро, это означает, что ему нужно, чтобы список прокручивался гораздо быстрее. Инерция сдвига должна позволить списку продолжать прокручиваться даже тогда, когда пользователь убирает палец с экрана. Это напоминает вращающееся по инерции колесо.

Манипуляции способны генерировать большие объемы информации в короткие промежутки времени, поэтому по умолчанию они отключены. Для того чтобы элемент управления мог реагировать на манипуляции, нужно настроить его свойство `ManipulationMode`. У сетки (элемент управления `Grid`) с именем `TouchGrid`, которая объявлена в файле MainPage.xaml, это свойство установлено в значение `All`. Возможные значения этого свойства заданы в перечислении `ManipulationModes`. Список манипуляций и их значения приведены в табл. 4.3.

Режим направляющих позволяет упростить взаимодействие пользователя с сенсорным интерфейсом в ситуациях, когда возможна прокрутка контента в различных направлениях. Проблема в том, что при горизонтальной или вертикальной прокрутке может проявляться эффект смещения изображения в направлении, отличном от направления прокрутки. Для того чтобы избежать этого ненужного эффекта, возможность подобного нежелательного смещения вообще блокируется до тех пор, пока палец находится в пределах некоторой области (направляющих). Любое перемещение

в пределах этой зоны ограничено либо горизонтальной, либо вертикальной прокруткой, в то время как перемещения вне этой зоны могут выполняться в любом направлении. Для того чтобы больше узнать об использовании направляющих, обратитесь к описанию операции сдвига на странице http://msdn.microsoft.com/ru-ru/library/windows/apps/hh465310.aspx.

Таблица 4.3. Значения из перечисления ManipulationModes

Значение	Описание
None	Не отслеживать события манипуляций
TranslateX	Разрешить манипуляции, которые сдвигают объект по оси X
TranslateY	Разрешить манипуляции, которые сдвигают объект по оси Y
TranslateRailsX	Разрешить манипуляции, которые сдвигают объект по оси X с использованием режима направляющих
TranslateRailsY	Разрешить манипуляции, которые сдвигают объект по оси Y с использованием режима направляющих
Rotate	Разрешить манипуляции, которые поворачивают объект
Scale	Разрешить манипуляции, которые масштабируют целевой объект
TranslateInertia	Включить применение инерции к жестам сдвига
RotateInertia	Включить применение инерции к жестам поворота
ScaleInertia	Включить применение инерции к жестам масштабирования
All	Разрешить все виды манипуляций и всех видов инерции
System	Зарезервировано

Вы можете настроить режим обработки манипуляций в XAML, задав необходимые параметры при описании элемента, как это сделано в примере. Если вам нужно совместить обработку нескольких манипуляций, требуется задать соответствующие параметры программно. В следующем примере манипуляции масштабирования и поворота без инерции совмещены:

```
TouchGrid.ManipulationMode = ManipulationModes.Rotate |
  ManipulationModes.Scale;
```

Отслеживание манипуляций выполняется путем обработки события `ManipulationDelta`. Это событие вызывается при выполнении жеста и пре-

доставляет данные о нескольких контрольных точках. При масштабировании, вращении и сдвиге (перетаскивании) объекта можно отслеживать как общие результаты изменения (то есть те изменения, которые произошли с момента начала манипуляции), так и небольшие изменения, произошедшие с момента предыдущего вызова соответствующего события. В нашем примере к элементу `Grid` применена трансформация `CompositeTransform`. Она позволяет осуществлять операции масштабирования, вращения и перетаскивания. Масштабирование и перетаскивание применяются в ходе манипуляции постепенно, а вращение — на основе общего результата манипуляции:

```
Transformation.ScaleX *= e.Delta.Scale;
Transformation.ScaleY *= e.Delta.Scale;
Transformation.TranslateX += e.Delta.Translation.X;
Transformation.TranslateY += e.Delta.Translation.Y;
Transformation.Rotation += e.Delta.Rotation;
```

Этот пример довольно прост, так как в нем результаты манипуляций напрямую применяются к сетке. Работая с приложением в примере, вы увидите, что оно хорошо реагирует на вращение, смещение, масштабирование, перетаскивание. Учитывается при обработке этих жестов и инерция. Если вы выполните быстрый жест сдвига, то изображения, которые находятся в сетке, сместятся за пределы видимой области элемента. Для того чтобы вернуть сетку в исходное состояние, выполните жест двойного касания экрана. Обработчик события, вызываемого этим жестом, очистит или сбросит все данные манипуляций:

```
Transformation.ScaleX = 1.0;
Transformation.ScaleY = 1.0;
Transformation.TranslateX = 0;
Transformation.TranslateY = 0;
Transformation.Rotation = 0;
```

Вы можете использовать данные, которые предоставляют объекты манипуляций, чтобы реализовывать с их помощью любое необходимое взаимодействие с приложением независимо от того, связано ли оно с непосредственным управлением элементами, изображенными на экране, либо выполнением любых других действий. Важно, чтобы вы предоставили пользователю аналогичные возможности и для других способов взаимодействия с приложением. Например, масштабирование объектов при работе с мышью выполняют с помощью ее колеса. Реализация той же функциональности с помощью клавиатуры обычно заключается в использовании сочетаний клавиши Ctrl совместно с клавишами + и – соответственно для увеличения и уменьшения масштаба.

Поддержка мыши

В данном примере поддержка мыши реализована посредством реакции на событие `PointerWheelChanged`. Это еще одно «абстрактное» событие, хотя оно, вероятнее всего, происходит в ответ на вращение колеса мыши. На самом деле то свойство, которое проверяют для получения сведений о направлении вращения колеса, имеет в имени слово «mouse» («мышь»). Взгляните на код обработчика события `TouchGrid_PointerWheelChanged_1` в файле MainPage.xxaml.cs, и вы увидите код, который проверяет свойство `MouseWheelDelta`, чтобы определить, следует ли увеличивать или уменьшать масштаб в ответ на вращение колеса:

```
var factor = e.GetCurrentPoint((UIElement)sender)
  .Properties.MouseWheelDelta > 0
  ? 0.1 : -0.1;
Transformation.ScaleX += factor;
Transformation.ScaleY += factor;
```

Если вы пользуетесь мышью или исполняете программу на имитаторе в режиме поддержки мыши, то обнаружите, что вращение колеса мыши приводит к тем же результатам, что и жест изменения масштаба. Для того чтобы быть более последовательным в реализации подобного взаимодействия с элементом управления `SemanticZoom`, вы, вероятно, решите применять подобный эффект лишь в том случае, когда пользователь будет прокручивать колесо мыши, держа нажатой клавишу Ctrl. Конечно, важно предоставить также возможность делать то же самое с помощью клавиатуры. В следующем разделе рассказано, как добиться этого для случая изменения масштаба.

Поддержка клавиатуры

Поддержку клавиатуры можно реализовать, используя события `KeyDown` и `KeyUp`. Одновременно могут быть нажаты несколько клавиш, поэтому комбинация этих событий позволяет гибко отслеживать нужные комбинации клавиш. В примере Touch можно изменять масштаб изображения, удерживая нажатой клавишу Ctrl и нажимая клавишу + или –.

Первый шаг в организации поддержки событий клавиатуры заключается в ожидании загрузки элемента управления `ListBox` и установке на него фокуса, чтобы он мог прослушивать события нажатия клавиш:

```
private void EventList_Loaded_1(object sender, RoutedEventArgs e)
{
  EventList.Focus(FocusState.Programmatic);
}
```

СОВЕТ

Для того чтобы события клавиатуры вызывались для некоторого элемента, к которому они привязаны, нужно, чтобы этот элемент имел фокус ввода. Обычно элемент приобретает фокус ввода либо тогда, когда на нем щелкают мышью, либо когда переходят к нему по нажатию клавиши Tab или клавиш со стрелками. Фокус ввода не устанавливается автоматически. Так как на элемент Grid, который используется в данном примере, фокус нельзя установить программно, события клавиатуры привязаны к элементу управления ListBox. Для того чтобы задавать события непосредственно для элемента Grid, можно создать нестандартный элемент управления (класс Control или UserControl), который в качестве основного элемента будет содержать сетку (элемент Grid). Класс Control поддерживает метод Focus. Он позволяет программно устанавливать фокус на элемент управления, реализованный с его помощью, для организации прослушивания событий нажатия клавиш. Этот метод подходит для любых базовых элементов управления, которые не поддерживают программную установку фокуса ввода.

Далее нужно отслеживать состояние клавиши Ctrl. Когда клавиша нажата, в обработчике события `KeyDown` устанавливается соответствующий флаг:

```
if (e.Key.Equals(VirtualKey.Control) && !_isCtrlKeyPressed)
{
  _isCtrlKeyPressed = true;
  AddWithFocus("Ctrl Key pressed.");
}
```

Когда клавиша отпущена, флаг сбрасывается:

```
private void EventList_KeyUp_1(object sender, KeyEventArgs e)
{
  if (e.Key.Equals(VirtualKey.Control))
  {
    _isCtrlKeyPressed = false;
    AddWithFocus("Ctrl key released.");
  }
}
```

И наконец, если в то время, когда нажата клавиша Ctrl, пользователь нажимает клавишу + или –, выполняется масштабирование изображения. В листинге 4.2 показан весь код обработчика события `KeyDown`.

Листинг 4.2. Код обработчика события, реализующего поддержку клавиатуры

```
private void EventList_KeyDown_1(object sender, KeyEventArgs e)
{
  if (e.Key.Equals(VirtualKey.Control) && !_isCtrlKeyPressed)
  {
    _isCtrlKeyPressed = true;
    AddWithFocus("Ctrl Key pressed.");
  }
  else if (_isCtrlKeyPressed)
  {
    var factor = 0d;
    if (e.Key.Equals(VirtualKey.Add))
    {
      factor = 0.1;
    }
    else if (e.Key.Equals(VirtualKey.Subtract))
    {
      factor = -0.1;
    }
    Transformation.ScaleX += factor;
    Transformation.ScaleY += factor;
  }
}
```

Теперь приложение поддерживает все варианты ввода: сенсорный ввод, а также ввод посредством мыши и клавиатуры.

Визуальная обратная связь

Компания Microsoft рекомендует, чтобы разработчики обеспечивали визуальную обратную связь для всех видов сенсорного взаимодействия, реализованных в их приложениях. Визуальная обратная связь показывает пользователю, какой именно функциональности можно ожидать от приложения, помогает обнаружить в интерфейсе приложения контент, с которым можно взаимодействовать. Все стандартные элементы управления в Windows 8 поддерживают визуальную обратную связь без дополнительных усилий со стороны разработчика. Если же вы создаете нестандартный элемент управления, то должны сами позаботиться о том, чтобы он предоставлял пользователю визуальную обратную связь.

Вот основные положения руководства по организации визуальной обратной связи при сенсорном взаимодействии с приложением:

- Все элементы управления следует снабдить механизмом обратной связи.
- Элемент управления должен обеспечивать обратную связь даже при кратком контакте с ним. Это помогает пользователю понять, что сенсорный экран работоспособен и что пользователь не промахнулся мимо цели касания. Кроме того, это может показать, что некий элемент не поддерживает сенсорное взаимодействие.
- Визуальная обратная связь должна срабатывать немедленно.
- Визуальная обратная связь должна быть ограничена только тем элементом, которого касается пользователь.
- Визуальная обратная связь не должна отвлекать пользователя от намеченных им действий.
- Визуальная обратная связь не должна включаться при перетаскивании или прокрутке элемента.
- Во время манипуляций следует обеспечить «прилипание» целевых объектов к пальцу пользователя.
- Если целевой объект не перемещается, следует показать, например, пунктирной линией, что манипуляция, которую выполняет пользователь, относится именно к этому объекту.

Когда пользователь касается UI-элемента и удерживает на нем палец, следует во время касания показывать контекстную информацию. При коротком удержании, длящемся менее 200 миллисекунд, следует показать простую всплывающую подсказку с коротким пояснением, касающимся объекта. Если же пользователь удерживает палец дольше чем 2 секунды, нужно показать всплывающее окно с более подробной информацией или подробный список доступных команд.

Для того чтобы увидеть пример подобного взаимодействия, коснитесь какой-нибудь плитки на начальном экране и удерживайте на ней палец. Для некоторых позиций появится краткая всплывающая подсказка с описанием плитки. Кроме того, вы заметите, что плитка слегка «вжимается» вглубь экрана. Подобная обратная связь показывает, как можно выделить плитку с помощью жеста скольжения поперек.

Теперь коснитесь плитки, но не удерживайте на ней палец, а слегка переместите его вниз (под прямым углом к направлению прокрутки начального экрана). Появится серый флажок выделения. Переместите плитку дальше, и флажок станет четче. Когда вы отпустите палец, плитка окажется выделенной. Повторите ту же операцию, чтобы снять выделение. Именно так, за счет скольжения поперек, можно выделять объекты. При этом элементы

управления самостоятельно предоставляют визуальные подсказки, которые показывают пороговые значения для действий. Кроме того, вы можете начать перетаскивать плитку, а затем вернуть ее туда же, где она была, при этом она не будет выделена.

И наконец, коснитесь плитки и начните перемещать ее, но в этот раз переместите ее дальше того места, по достижении которого осуществлялось ее выделение, на что указывало закрашивание флажка. Очевидно, при этом вы пересечете порог выделения. Когда это произойдет, плитка «приклеится» к вашему пальцу, весь начальный экран слегка уменьшится, и остальные плитки начнут смещаться, предоставляя место для той плитки, которую вы перетаскиваете по экрану. В этом режиме вы можете переместить плитку в новое место на начальном экране. Описанные действия иллюстрирует рис. 4.5.

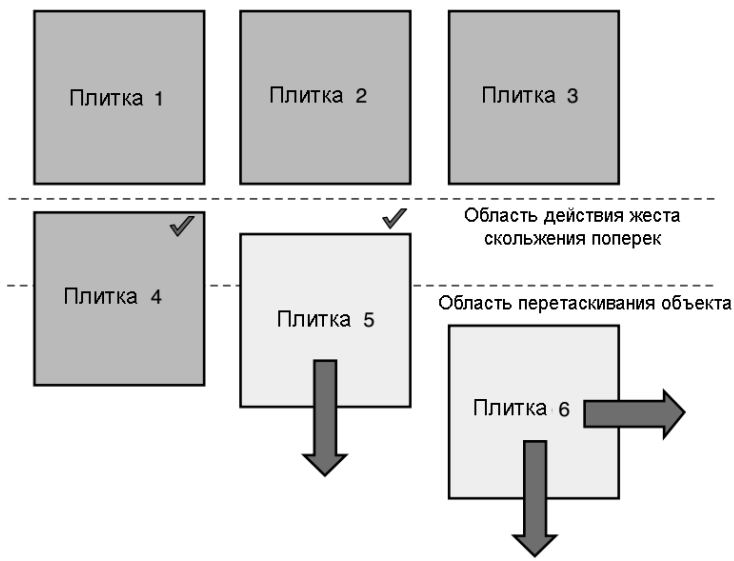


Рис. 4.5. Скольжение по диагонали для выделения и перетаскивание

В примере плитка 4 уже выделена. Плитку 5 перемещают. Так как она не пересекла порог выделения, она перемещается лишь вертикально, на ней выводится символ флажка как подсказка о том, что таким способом плитку можно выделить. Если эту плитку переместить назад, она встанет на прежнее место и не будет выделена. Плитку 6 перетащили за пределы порога выделения, далее ее можно перемещать в любом направлении, чтобы расположить ее на новом месте.

Организация целей касания

Организация целей касания связана с созданием UI-элементов, максимально адаптированных для сенсорного взаимодействия. Для того чтобы создавать удобные интерфейсы, рассчитанные на сенсорное управление, нужно учитывать несколько факторов. Первый заключается в том, чтобы разрабатываемые целевые элементы были достаточно велики для того, чтобы с ними удобно было работать. В слишком маленькие цели касания тяжело попасть пальцем, отсюда возрастает количество ошибок при работе с программой (не говоря уже о недовольстве пользователя). Руководство по целям касания рекомендует, чтобы размер короткой стороны целевого элемента был не меньше 9 миллиметров (это 48 пикселов на дисплее с плотностью 135 пикселов на дюйм). Рекомендуется также, чтобы расстояние между целями касания было не меньше чем 2 миллиметра (около 10 пикселов).

Если целевой элемент перемещается меньше чем на 2,7 миллиметра (примерно 14 пикселов), следует расценивать это как жест касания. Перемещение на большее расстояние следует обрабатывать как перетаскивание элемента. Если цель касания для выделения поддерживает скольжение поперек, выделение выполняется тогда, когда объект пересекает эту границу. Выделение, в свою очередь, переходит в перетаскивание, когда пользователь перемещает элемент на 11 миллиметров (примерно 60 пикселов).

Помимо понятия *визуальной цели касания* (visual target), которая представляет собой видимый элемент, существует понятие *фактической цели касания* (actual target). Фактическая цель касания — это область вокруг элемента, которая может реагировать на касания, нацеленные на данный элемент. Рекомендовано, чтобы визуальный размер цели составлял примерно 60 % от ее фактического размера. При взаимодействии с фактическими целями касания может выводиться подсказка. Для того чтобы увидеть, как этот механизм работает, перейдите на начальный экран и взгляните на информацию о вашей учетной записи (рис. 4.6).

Рис. 4.6. Сведения об учетной записи

Изображение и текст — это визуальная цель касания. Теперь щелкните в любом месте внутри воображаемого прямоугольника, который достаточно велик, чтобы вместить в себя и текст, и изображение. Сведения об учетной записи будут выделены, а прямоугольник выделения покажет размеры фактической цели касания (рис. 4.7), щелчок в пределах которой откроет контекстное меню учетной записи.

Рис. 4.7. Сведения об учетной записи и прямоугольник, показывающий размеры фактической цели касания

Другой важный аспект, который имеет отношение к целям касания, заключается в минимизации риска критических ошибок. Элементы, ошибочное использование которых может привести к неприятным последствиям, следует располагать отдельно от элементов, требуемых при обычной работе. Это позволит избежать случайной активации первых. Например, если вам нужно разместить в интерфейсе приложения команды добавления, модификации и удаления, то команды добавления и модификации можно разместить рядом, а команду удаления — на некотором расстоянии от них, чтобы снизить вероятность того, что пользователь случайно ее выполнит.

Контекстные меню

Контекстные меню — это небольшие меню, которые обычно содержат команды для работы с буфером обмена (вырезания, копирования и вставки) или нестандартные команды для объектов, не поддерживающих выделение. В такое меню включают короткий список команд, как правило, не больше пяти. Рекомендации по работе с контекстными меню вы можете прочитать на странице http://msdn.microsoft.com/ru-ru/library/windows/apps/hh465308.aspx.

В этих рекомендациях написано, что у команд контекстных меню должны быть короткие имена, начинающиеся с прописной буквы (то есть «Очистить список», но не «Очистить Список»). Кроме того, рекомендуется использовать разделители для группировки взаимосвязанных команд. Команды должны соответствовать контексту выполняемой операции, их следует располагать в меню в соответствии с их значимостью, самые важ-

ные команды — в верхней части меню. Если вы включаете в меню команды для работы с буфером обмена, следует всегда использовать для них следующие имена и порядок расположения:

- Cut (Вырезать);
- Copy (Копировать);
- Paste (Вставить).

Старайтесь, чтобы контекстные меню располагались как можно ближе к объекту, с которым они связаны, так как их часто вызывают жестом касания или удержания. В примере Touch у элемента `LixtBox`, который выводит сведения о сенсорных событиях, есть контекстное меню.

Обработчик события содержит небольшой объем программного кода, который создает объект `PopupMenu`, добавляет в созданное контекстное меню команду и затем ожидает реакции пользователя. Контекстное меню можно создать и в XAML-разметке, используя свойство `Button.ContextMenu`. Содержимое свойства включает в себя описание объекта `ContextMenu` с набором команд, представленных элементами управления `MenuItem`. Меню может автоматически скрываться, когда пользователь выбирает команду или убирает с меню фокус ввода (например, касаясь другой области в приложении или щелкая на ней мышью):

```
var contextMenu = new PopupMenu();
contextMenu.Commands.Add(new UICommand("Clear list",
  args => _events.Clear()));
var dismissed = await contextMenu.ShowAsync(
  e.GetPosition(EventList));
```

Конструктору объекта `UICommand` передается короткое имя команды и делегат, который будет вызван при выборе пользователем соответствующего пункта меню. Вы можете разделять команды меню, добавляя объект `UICommandSeparator` в их список:

```
contextMenu.Commands.Add(new UICommandSeparator());
```

Команды в меню появляются в соответствии с порядком их добавления. Вы можете воспользоваться значением, которое возвращает метод `ShowAsync`, чтобы выяснить, почему меню было закрыто. Некоторые элементы управления предоставляют контекстные меню автоматически. Например, для выделенного текста предоставляется набор команд для работы с буфером обмена. У вашего приложения наверняка будут команды, которые не вписываются в узкие рамки рекомендаций, касающихся контекстных меню. Вероятнее всего, эти команды можно будет разместить в панели приложения.

Панель приложения

Панель приложения (application bar) — это специальный элемент управления, который выводит команды на экран тогда, когда они нужны пользователю. По умолчанию панель приложения скрывается, когда ею не пользуются. Для того чтобы ее вывести, пользователь должен либо выполнить жест скольжения от верхнего или нижнего края экрана к центру, либо щелкнуть в рабочем поле приложения правой кнопкой мыши. Панель приложения можно выводить на экран и программным путем. Она предназначена для размещения контекстных команд. Это могут быть как глобальные команды, например команда возвращения на домашнюю страницу приложения, так и команды, имеющие отношение к выделенному элементу.

В примере Windows8Application показано, как использовать элемент управления `UserControl` для создания панели приложения. Благодаря этой панели обеспечивается централизованное расположение команд и предотвращается дублирование кода и XAML-разметки (вы можете, если хотите, поместить на каждую страницу собственную панель приложения). Панель приложения может располагаться в верхней или нижней части дисплея либо и там, и там. Когда пользователь при открытой панели приложения проводит по экрану пальцем или выбирает на панели команду, она автоматически закрывается. Панель приложения представлена в системе элементом управления `AppBar`.

На рис. 4.8 показана панель приложения главной страницы в ситуации, когда на странице нет выделенных элементов. Кнопки Add (добавить) и Home (домой) недоступны, так как на странице нет выделенной группы, и пользователь уже находится на домашней странице. Панель приложения (в данном случае она снизу) задана с помощью элемента `Page.BottomAppBar`. Также вы можете использовать элемент `Page.TopAppBar` для описания панели приложения, которая будет располагаться в верхней части экрана.

Любая панель приложения должна содержать элемент управления `AppBar`, который, в свою очередь, содержит XAML-разметку, описывающую макет для размещения команд приложения. Вот соответствующая разметка из файла GroupedItemsPage.xaml:

```
<Page.BottomAppBar>
  <AppBar x:Name="AppBar" Margin="10,0,10,0">
    <local:ApplicationCommands x:Name="AppBarCommands"/>
  </AppBar>
</Page.BottomAppBar>
```

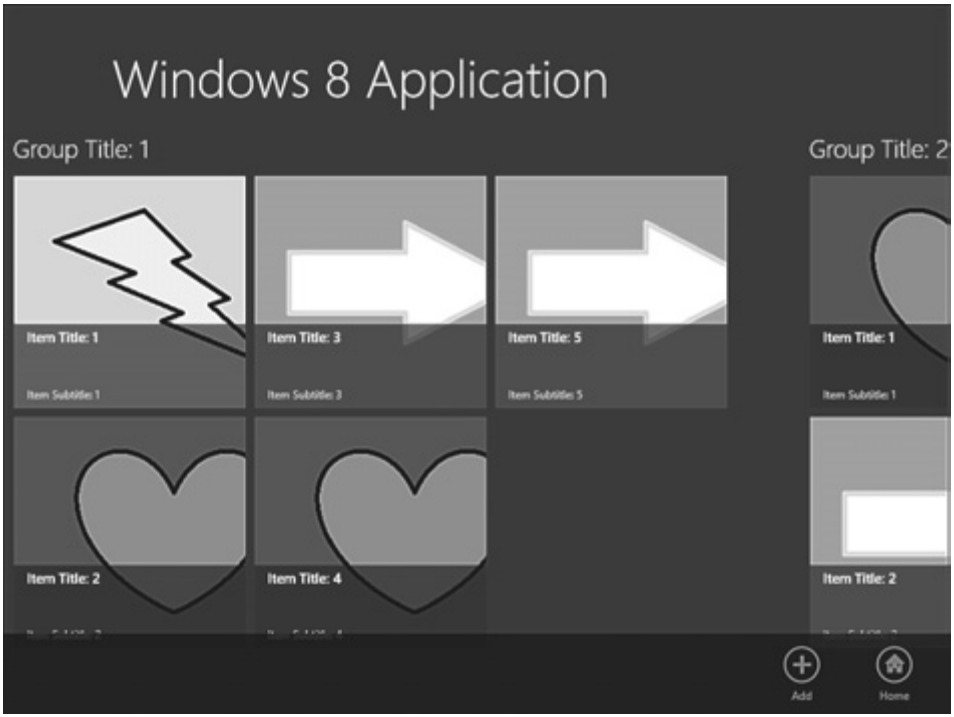


Рис. 4.8. Панели приложения с недоступными командами

XAML-разметка ссылается на многократно используемый элемент управления `UserControl`. Для того чтобы увидеть описание макета панели приложения, откройте файл ApplicationCommads.xaml. Вы увидите команды, которые в окне конструктора располагаются на сером фоне. Команда Delete (удалить) имеет вид кнопки, которая расположена в левой части панели. Команды Add (добавить) и Home (домой) расположены в правой части панели. Их расположение определяет сетка (элемент `Grid`), на основе которой построен макет.

Вы можете заметить, что команды описаны с помощью статических ресурсов:

```
<Button x:Name="Delete" HorizontalAlignment="Right"
  Style="{StaticResource DeleteAppBarButtonStyle}"
  Click="Delete_Click_1"/>
```

Эти ресурсы заданы в файле Common/StandardStyles.xaml, который автоматически генерируется на основе шаблона проекта. Стили содержат описания наиболее часто используемых команд. Каждый стиль предоставляет значок команды и описание. Например, следующая XAML-разметка описывает команду Help (справка), причем все стили ориенти-

рованы на элемент управления Button (кнопка) и основаны на ресурсе AppBarButtonStyle:

```
<Style x:Key="HelpAppBarButtonStyle" ...>
  <Setter Property="AutomationProperties.Name" Value="Help"/>
  <Setter Property="Content" Value="&#xE11B;"/>
</Style>
```

Если вы не можете найти стиль, который содержит нужный вам значок, просто создайте новый ресурс. Не рекомендуется модифицировать ресурсы в файле StandardStyles.xaml, если вы не полностью понимаете смысл своих действий, так как это может привести к ошибкам при работе приложения. Для того чтобы обезопасить себя от подобных ошибок, можно добавить собственный словарь стилей и указать ссылку на него в файле App.xaml (см. файл MyStyles.xaml). Представим, что вы создаете приложение для чтения новостных статей и хотите добавить в него команду для изменения размера шрифта. Для начала определим стиль новой команды:

```
<Style x:Key="FontAppBarButtonStyle" TargetType="Button"
  BasedOn="{StaticResource FontBarButtonStyle}">
</Style>
```

Вы можете задать для команды имя и свойства автоматизации (AutomationProperties). AutomationProperties — это уникальный идентификатор, который помогает организовать UI-автоматизацию. Данная возможность используется в различных вариантах тестирования и для поддержки специальных возможностей.

```
<Setter Property="AutomationProperties.AutomationId"
  Value="FontAppBarButton"/>
<Setter Property="AutomationProperties.Name" Value="Font"/>
```

Последний шаг заключается в задании значка. Если вы умеете рисовать или у вас есть дизайнер, значок можно создать самостоятельно. Однако обычно целесообразнее воспользоваться стандартными значками Windows 8, которые имеются в шрифте Segoe UI Symbol. Для того чтобы просмотреть состав символов этого шрифта, перейдите на начальный экран и наберите с клавиатуры поисковый запрос charmap. С панели результатов поиска запустите приложение charmap.exe. Это инструмент, который предназначен для просмотра шрифтов, установленных в системе. Выберите в списке доступных шрифтов шрифт Segoe UI Symbol и прокрутите список символов ближе к концу. Вы увидите довольно много значков. На рис. 4.9 показан значок, который, вероятно, лучше всего подходит для команды управления шрифтами, которую мы хотим создать.

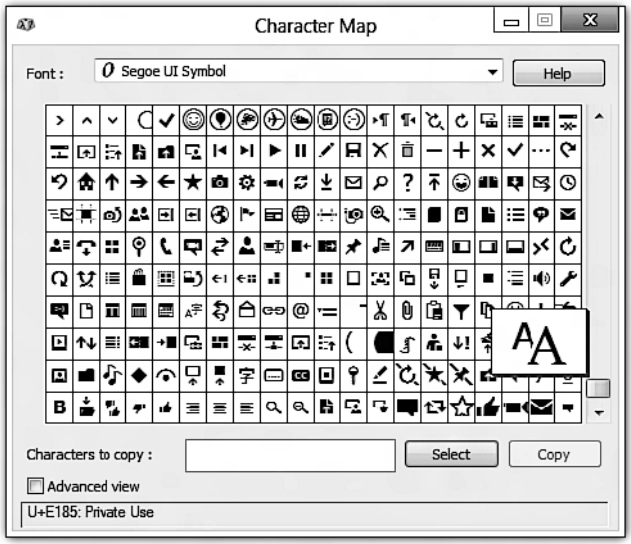


Рис. 4.9. Выбор нужного значка с помощью приложения Character Map

Обратите внимание на код символа в нижней части окна. Вы можете использовать этот код, чтобы задать необходимый символ в приложении, просто скопировав ту его часть, которая расположена после знака «плюс» (+):

```
<Setter Property="Content" Value="&#xE185;"/>
```

Теперь у вас есть стиль, который можно вызвать из любого места приложения.

Программный код для панели приложения делает команды доступными или недоступными, основываясь на текущем выделении. Вы можете изучить файл ApplicationCommands.xaml.cs, чтобы это увидеть. Например, команда удаления должна появляться только тогда, когда выделен некий элемент:

```
var selected = App.CurrentItem;
Delete.Visibility = selected == null ?
  Visibility.Collapsed : Visibility.Visible;
```

На рис 4.10 вы можете видеть на панели приложения кнопку для команды Delete (удалить). Обратите внимание, что команды Add (добавить) на панели нет, так как она не имеет смысла в контексте отдельного элемента.

Реализация команды Delete (удалить) выглядит несколько сложнее. Для того чтобы обезопасить пользователя от случайного удаления элемента, перед выполнением операции у него запрашивается подтверждение. Если он подтвердил удаление элемента и при этом открыта страница просмо-

Рис. 4.10. Доступные команды на панели приложения

тра подробных сведений об этом элементе, команда осуществит возврат к предыдущей странице, так как текущая страница после удаления элемента перестает существовать. Диалоговое окно для запроса подтверждения можно настроить следующим образом:

```
var msg = new Windows.UI.Popups.MessageDialog("Confirm Delete",
  string.Format("Are you sure you wish to delete the item \"{0}\"",
                App.CurrentItem.Title));
```

Команда OK в этом окне связана с делегатом, который удаляет элемент из группы и проверяет текущую страницу, чтобы определить, нужно ли переходить на предыдущую страницу. Для того чтобы увидеть, как выполняется анализ страницы, вы можете взглянуть в код файла App.xaml.cs:

```
App.CurrentGroup.Items.Remove(App.CurrentItem);
if (App.NavigatedPage == typeof(ItemDetailPage))
{
  ((Frame)Window.Current.Content).GoBack();
}
```

Команда Cancel (отмена) диалогового окна закрывает панель приложения, делегат ей не передается:

```
msg.Commands.Add(new Windows.UI.Popups.UICommand("Cancel"));
```

На рис. 4.11 вы можете видеть диалоговое окно, запрашивающее у пользователя подтверждение на выполнение операции.

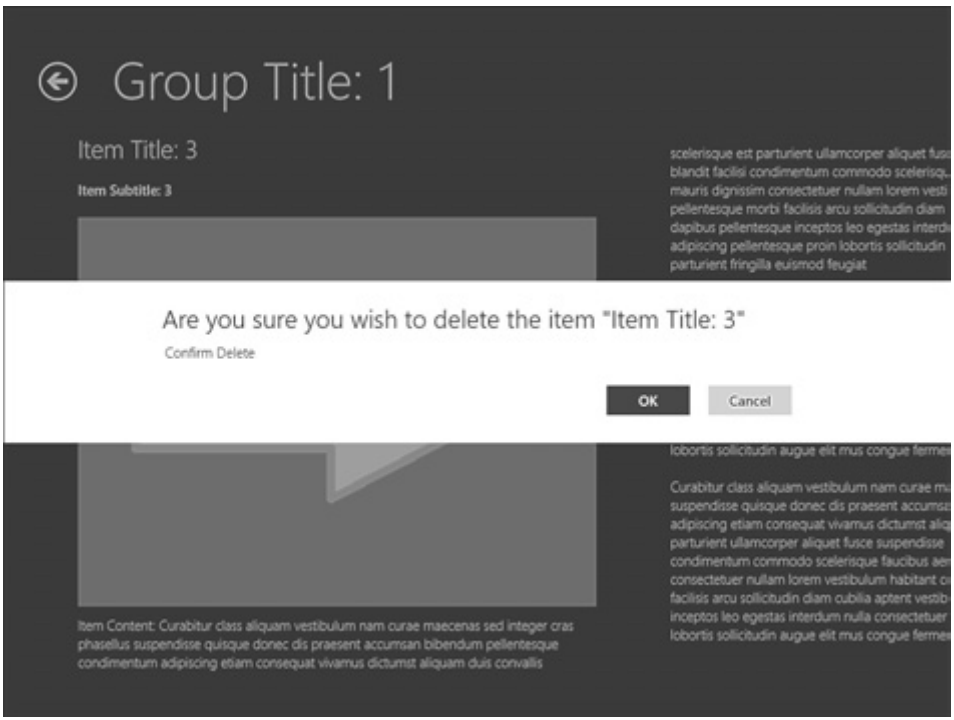


Рис. 4.11. Диалоговое окно, запрашивающее у пользователя подтверждение на удаление объекта

Элемент управления `AppBar`, представляющий панель приложения, можно программно показывать и скрывать, манипулируя его свойством `IsOpen`. Стандартный режим работы панели приложения заключается в том, что она исчезает, когда пользователь взаимодействует с областями приложения, которые находятся за ее пределами. Если установить свойство `IsSticky` панели приложения в значение `true`, она будет оставаться на экране до тех пор, пока пользователь явным образом не закроет ее, щелкнув правой кнопкой мыши, нажав комбинацию клавиш Ctrl+Z или выполнив жест скользящего движения (swipe) на сенсорном экране.

СОВЕТ

Если вы взглянете на элементы управления панели приложения в конструкторе, то увидите, что кнопки отображаются в больших прямоугольниках. Это стандартный режим просмотра элементов управления при разработке. Вы можете изменить этот режим, используя в XAML расширение d:. Например, если вы хотите, чтобы элемент управления имел бы в конструкторе определенную ширину, используйте команду d:DesignWidth="1024" после объявления xmlns:d. Если нужно, чтобы конструктор имитировал внешний вид планшетного компьютера, просто добавьте объявление d:ExtensionType="Page". В результате конструктор будет выводить окно в виде планшета, также он даст возможность использовать различные элементы управления и режимы просмотра. Эта небольшая хитрость позволяет расширить функциональность, доступную при разработке приложения.

При расположении команд на панели приложения важно следовать рекомендациям. Вот основные положения этих рекомендаций:

- Глобальные команды следует размещать в правой части панели.
- Команды следует размещать, начиная с правой и левой частей панели приложения и добавляя новые команды в направлении центра панели.
- Похожие команды следует объединять в группы с использованием разделителей.
- Контекстные команды следует размещать ближе к левому краю панели и скрывать их, когда они больше не требуются.
- Команды, ошибочное использование которых может привести к неприятным последствиям, следует располагать на некотором расстоянии от других команд, чтобы снизить вероятность их случайного использования.
- Не размещайте на панели приложения команды настройки программы. Используйте для этого возможности чудо-кнопки Settings (параметры).
- Не применяйте панель приложения для реализации поиска, если только поиск не выполняется в пределах данных, видимых на странице. Используйте для поиска возможности чудо-кнопки Search (поиск).
- Не применяйте панель приложения для размещения команд отправки данных в другие приложения. Для подобных целей используйте возможности чудо-кнопки Share (общий доступ).
- Если на панели приложения расположено слишком много команд, ее структуру можно улучшить за счет всплывающих меню. Можно также распределить команды между верхней и нижней панелями, учитывая особенности их использования.

Последнюю версию рекомендаций, касающихся панелей приложения, можно найти на странице http://msdn.microsoft.com/en-us/library/windows/apps/hh465302.aspx.

Запустите приложение снова и попрактикуйтесь в добавлении и удалении элементов. Вы должны заметить одну полезную возможность, которую без усилий с вашей стороны предоставляет стандартный элемент управления `Grid` (сетка). Так, когда существующие в сетке элементы сдвигаются, их перемещение анимируется. Все это происходит автоматически, отражая результат воздействия на коллекцию, в которой хранятся элементы, выводимые на экран.

Возможности панели приложения не ограничиваются размещением на ней значков и команд. Может иметь смысл вывод на панели приложения и других данных, например индикаторов (если у вас есть телефон под управлением операционной системы Windows Phone, вам знакомы индикаторы на панели приложения, которые показывают уровень заряда батареи и силу сигнала сотовой сети). Также на панели приложения можно размещать эскизы изображений или страниц. Взгляните на окно Internet Explorer на рис. 4.12. Здесь панель приложения служит для переключения между существующими вкладками или для добавления новых.

Рис. 4.12. Креативное использование панели приложения в Internet Explorer

Значки и экраны-заставки

Вы можете заметить, что при запуске приложения Windows8Application появляется созданный специально для него зеленый экран-заставка. Его использование преследуют две цели. Во-первых, по его внешнему виду можно судить, что здесь руку приложил не дизайнер, а разработчик. Во-вторых, он показывает, как можно настраивать различные значки и логотипы, имеющие отношение к вашему приложению.

Для настройки плиток и экрана-заставки приложения выполните двойной щелчок на файле Package.appxmanifest, чтобы открыть его в редакторе манифеста приложения. Значки и экран-заставку можно настроить на вкладке Application UI, показанной на рис. 4.13. Здесь можно задать особенности вывода названия приложения и цвета, которые при этом используются. Вы можете задать логотипы для плиток различных видов, например маленький значок (Small logo) применяется при работе с начальным экраном в режиме уменьшенного масштаба. Также здесь можно настроить изображение и фоновый цвет для экрана-заставки.

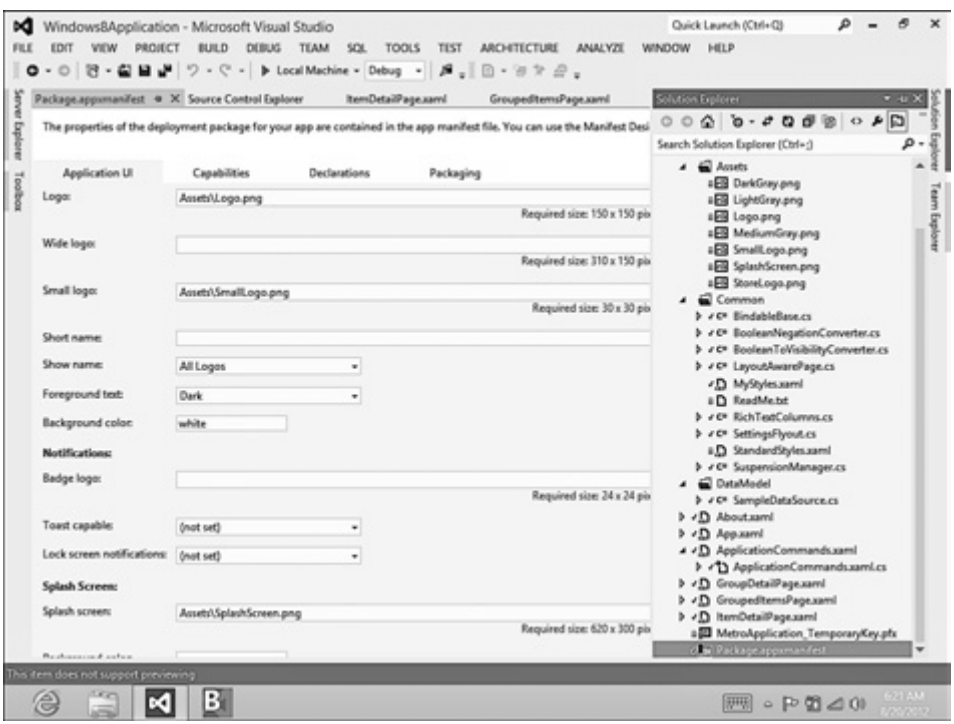


Рис. 4.13. Вкладка для редактирования параметров интерфейса приложения в редакторе манифеста

В данном диалоговом окне приведены советы, которые касаются подходящих размеров различных графических элементов. О том, как создать нестандартный экран-заставку, который может включать различные наборы элементов, вы узнаете из главы 5. О плитках и всплывающих уведомлениях вы узнаете в главе 7.

Страница информации о программе

Для того чтобы предоставить пользователю доступ к параметрам приложения, к настройке предпочтений и к странице информации о программе, следует задействовать возможности чудо-кнопки Settings (параметры). Если вы при запущенном приложении воспользуетесь чудо-кнопкой Settings, на ее панели всегда будет содержаться команда Permissions (разрешения). Панель этой команды содержит название приложения, его версию и издателя, а также список разрешений, предоставленных приложению. Вспомните, как в главе 2 мы объявляли разрешение на использование веб-камеры.

Первый шаг в разработке страниц, которые планируется выводить на экран в качестве части чудо-кнопки Settings, заключается в создании класса, позволяющего странице выдвигаться с края экрана. Это так называемый *всплывающий элемент* (flyout). Его нет среди встроенных элементов управления, доступных XAML-приложениям. Создать этот класс довольно просто, заключив элемент управления в контейнер `Popup`, чтобы он мог перекрывать окно приложения. В папке Common проекта приложения Windows8Application вы можете найти определение класса `SettingsFlyout`, которое приведено в листинге 4.3. Существуют два стандартных размера для всплывающих элементов, которые отображаются в области чудо-кнопок: узкий (346 пикселов в ширину) и широкий (646 пикселов в ширину).

Листинг 4.3. Класс SettingsFlyout

```
class SettingsFlyout
{
  private const int _width = 346;
  private Popup _popup;

  public void ShowFlyout(UserControl control)
  {
    _popup = new Popup();
    _popup.Closed += OnPopupClosed;
    Window.Current.Activated += OnWindowActivated;
    _popup.IsLightDismissEnabled = true;
    _popup.Width = _width;
    _popup.Height = Window.Current.Bounds.Height;
```

```
    control.Width = _width;
    control.Height = Window.Current.Bounds.Height;

    _popup.Child = control;
    _popup.SetValue(Canvas.LeftProperty,
    Window.Current.Bounds.Width - _width);
    _popup.SetValue(Canvas.TopProperty, 0);
    _popup.IsOpen = true;
  }

  private void OnWindowActivated(object sender,
    Windows.UI.Core.WindowActivatedEventArgs e)
  {
    if (e.WindowActivationState ==
      Windows.UI.Core.CoreWindowActivationState.Deactivated)
    {
      _popup.IsOpen = false;
    }
  }

  void OnPopupClosed(object sender, object e)
  {
    Window.Current.Activated -= OnWindowActivated;
  }
}
```

Для того чтобы познакомиться с дизайном страницы, обратитесь к файлу About.xaml. Страницы для размещения на панели параметров имеют белый фон. Сетке (элементу `Grid`) задан переход, который обеспечивает анимацию текста, позволяя ему «выдвигаться» вместе с элементом управления. Особый стиль `SettingsBackButtonStyle` является копией стиля `BackButtonStyle`, который поставляется с шаблоном; он модифицирован для вывода на белом фоне.

На странице присутствуют две команды. Первая закрывает родительский элемент (класс `SettingsFlyout`) и возвращается к панели параметров приложения. Для реализации этой команды нужно использовать директиву `using` для пространства имен `Windows.UI.ApplicationSettings`:

```
private void Button_Click_1(object sender, RoutedEventArgs e)
{
  if (this.Parent.GetType() == typeof(Popup))
  {
    ((Popup)this.Parent).IsOpen = false;
  }
  SettingsPane.Show();
}
```

Вторая команда связана с кнопкой-гиперссылкой (элемент управления HyperlinkButton), щелчок на которой открывает страницу моего блога. Данная команда использует класс Launcher, чтобы обратиться к интернет-ресурсу по URI (Uniform Resource Identifier — унифицированный идентификатор ресурса). Это позволяет платформе Windows 8 определить, какое приложение зарегистрировано для обработки конкретного URI-идентификатора (в данном случае это, вероятнее всего, Internet Explorer 10), а затем – либо сразу передать данному приложению этот URI-идентификатор, либо перед передачей сначала запустить приложение.

```
private async void HyperlinkButton_Click_1(object sender,
  RoutedEventArgs e)
{
  await Windows.System.Launcher.LaunchUriAsync(
    new Uri("http://csharperimage.jeremylikness.com/"));
}
```

Последний шаг заключается в регистрации новой страницы для работы с чудо-кнопкой Settings. Когда приложение запущено, нужно обработать событие CommandsRequested объекта SettingsPane. Это событие вызывается, когда пользователь активирует чудо-кнопку Settings и создается список элементов ее панели. Регистрация события производится в файле App.xaml.cs:

```
SettingsPane.GetForCurrentView().CommandsRequested +=
  App_CommandsRequested;
```

Обработчик события работает с объектом Request, который содержит список команд. Вы можете добавлять в этот список собственные команды, чтобы они появлялись при вызове чудо-кнопки Settings:

```
var about = new SettingsCommand("about", "About", (handler) =>
{
  var settings = new SettingsFlyout();
  settings.ShowFlyout(new About());
});
args.Request.ApplicationCommands.Add(about);
```

Запустите приложение и протестируйте его новые возможности. Страница информации о программе (About) показана на рис. 4.14. Для того чтобы ее увидеть, выполните жест скольжения от правого края экрана к центру либо наведите указатель мыши на нижний правый угол экрана и держите его там до тех пор, пока не появится панель чудо-кнопок. Коснитесь чудо-кнопки Settings, и вы увидите, как в открывшейся панели в дополнение к стандартной команде Permissions появится команда About. Активируйте эту команду, и вы увидите, как на экране появляется новая страница. Ее появление сопровождается анимацией.

Рис. 4.14. Страница информации о программе (About)

Вы будете пользоваться похожей методикой при разработке реальных приложений для создания страниц настройки параметров и задания предпочтений пользователя. Больше о том, как хранить и считывать данные, вы узнаете в главе 5, а о работе с параметрами приложения — в главе 6.

Датчики

Приложения для Windows 8 могут обрабатывать не только те данные, которые поступают от сенсорного экрана. Многие устройства, работающие под управлением Windows 8, обладают специальными датчиками, которые предоставляют такие данные, как ориентация планшетного компьютера в пространстве или его GPS-координаты. Исполняемая среда Windows содержит особые прикладные программные интерфейсы, предназначенные для работы с этими датчиками. Их используют при создании приложений, которые реагируют на события, такие как встряхивания или поворот планшетного компьютера, либо предоставляют информацию, связанную с географическим положением пользователя. В этом разделе содержится краткий обзор прикладных программных интерфейсов для работы с датчиками.

Если у вас есть устройство, в которое интегрированы датчики, вы можете загрузить из центра разработки Microsoft примеры, иллюстрирующие работу с ними: http://code.msdn.microsoft.com/windowsapps/Windows-8-Modern-Style-App-Samples. На данной странице можно найти примеры, касающиеся каждого из описываемых в этом разделе датчиков. Ищите примеры, в названии которых есть слова Accelerometer (акселерометр), Gyrometer (гирометр), Sensor (датчик), Location (определение местоположения).

Акселерометр

Акселерометр (accelerometer) предоставляет сведения о смещении устройства относительно различных осей. API позволяет получить текущие сведения с датчика, вызывая соответствующее событие при изменении показаний. Событие предоставляет сведения об изменении положения относительно осей X, Y и Z. Код в листинге 4.4 демонстрирует получение показаний акселерометра.

Листинг 4.4. Чтение показаний акселерометра

```
async private void ReadingChanged(object sender,
  AccelerometerReadingChangedEventArgs e)
{
  await Dispatcher.RunAsync(CoreDispatcherPriority.Normal, () =>
  {
    AccelerometerReading reading = e.Reading;
    ScenarioOutput_X.Text = String.Format("{0,5:0.00}",
      reading.AccelerationX);
    ScenarioOutput_Y.Text = String.Format("{0,5:0.00}",
      reading.AccelerationY);
    ScenarioOutput_Z.Text = String.Format("{0,5:0.00}",
      reading.AccelerationZ);
  });
}
```

Подробнее о работе с акселерометром вы можете узнать на странице http://msdn.microsoft.com/ru-ru/library/windows/apps/windows.devices.sensors.accelerometer.aspx. Там же вы можете найти пример использования акселерометра.

Компас

Если в устройстве присутствует компас (compass), он предоставляет сведения об ориентации устройства относительно истинного севера (true north) или магнитного севера (magnetic north). *Истинный север* указывает на

географический северный полюс Земли, *магнитный север* — на магнитный северный полюс. Магнитный северный полюс не совпадает с географическим из-за изменений магнитного поля Земли. В листинге 4.5 представлен код для чтения показаний компаса.

Листинг 4.5. Чтение показаний компаса

```
async private void ReadingChanged(object sender,
  CompassReadingChangedEventArgs e)
{
  await Dispatcher.RunAsync(CoreDispatcherPriority.Normal, () =>
  {
    CompassReading reading = e.Reading;
    ScenarioOutput_MagneticNorth.Text =
      String.Format("{0,5:0.00}",
      reading.HeadingMagneticNorth);
    if (reading.HeadingTrueNorth != null)
    {
      ScenarioOutput_TrueNorth.Text =
        String.Format("{0,5:0.00}",
        reading.HeadingTrueNorth);
    }
    else
    {
      ScenarioOutput_TrueNorth.Text = "No data";
    }
  });
}
```

Подробности о работе с компасом вы можете найти на странице http://msdn.microsoft.com/ru-ru/library/windows/apps/windows.devices.sensors.compass.aspx.

Определение географического положения устройства

API для определения географического положения устройства (geolocation) собирает сведения из различных источников, чтобы предоставить информацию о примерном географическом положении устройства. Эти сведения могут поступать от поставщика местоположения Windows (Windows Location Provider), который использует комбинацию данных, полученных на основе Wi-Fi-триангуляции, на основе анализа IP-адреса либо на базе данных, полученных от специализированных датчиков, наподобие GPS-сенсора. API с помощью этих сведений пытается максимально точно вычислить текущее местоположение устройства.

В листинге 4.6 показан пример использования данного прикладного программного интерфейса для получения приблизительных данных о географической широте и долготе устройства.

Листинг 4.6. Получение сведений о географическом положении устройства

```
Geoposition pos = await _geolocator.GetGeopositionAsync()
     .AsTask(token);
ScenarioOutput_Latitude.Text = pos.Coordinate.Latitude.ToString();
ScenarioOutput_Longitude.Text = pos.Coordinate.Longitude.ToString();
ScenarioOutput_Accuracy.Text = pos.Coordinate.Accuracy.ToString();
```

Детали, касающиеся использования API для определения географического положения устройств, вы можете узнать на странице http://msdn.microsoft.com/ru-ru/library/windows/apps/windows.devices.geolocation.aspx.

Гирометр

Гирометр (gyrometer) — это датчик, который предоставляет сведения об угловой скорости устройства, о его вращении. Как и в случае с другими датчиками, существуют специальный прикладной программный интерфейс для получения показаний от гирометра, и событие, которое вызывается при изменении показаний. В листинге 4.7 показано, как интерпретировать данные, которые предоставляет это событие.

Листинг 4.7. Чтение показаний гирометра

```
async private void ReadingChanged(object sender,
  GyrometerReadingChangedEventArgs e)
{
  await Dispatcher.RunAsync(CoreDispatcherPriority.Normal, () =>
  {
    GyrometerReading reading = e.Reading;
    ScenarioOutput_X.Text = String.Format("{0,5:0.00}",
      reading.AngularVelocityX);
    ScenarioOutput_Y.Text = String.Format("{0,5:0.00}",
      reading.AngularVelocityY);
    ScenarioOutput_Z.Text = String.Format("{0,5:0.00}",
      reading.AngularVelocityZ);
  });
}
```

Подробности о работе с гирометром вы можете найти на странице http://msdn.microsoft.com/ru-ru/library/windows/apps/windows.devices.sensors.gyrometer.aspx.

Инклинометр

Инклинометр (inclinometer), или датчик наклона, предоставляет сведения об угле наклона устройства по осям X, Y и Z, в частности, данные о тангаже (pitch), крене (roll) и рыскании (yaw). Эти показатели позволяют понять, как именно устройство ориентировано относительно земли (или, точнее, направление воздействия на устройство силы тяжести). Их используют для определения величин поворотов и наклонов устройства. Объяснение понятий тангажа, крена и рыскания можно найти на веб-сайте NASA (http://www.grc.nasa.gov/WWW/K-12/airplane/rotations.html).

В листинге 4.8 представлен код для чтения показаний инклинометра.

Листинг 4.8. Чтение показателей инклинометра

```
async private void ReadingChanged(object sender,
  InclinometerReadingChangedEventArgs e)
{
  await Dispatcher.RunAsync(CoreDispatcherPriority.Normal, () =>
  {
    InclinometerReading reading = e.Reading;
    ScenarioOutput_X.Text = String.Format("{0,5:0.00}",
      reading.PitchDegrees);
    ScenarioOutput_Y.Text = String.Format("{0,5:0.00}",
      reading.RollDegrees);
    ScenarioOutput_Z.Text = String.Format("{0,5:0.00}",
      reading.YawDegrees);
  });
}
```

Подробнее о работе с инклинометром вы можете узнать на странице http://msdn.microsoft.com/ru-ru/library/windows/apps/windows.devices.sensors.inclinometer.aspx.

Датчик уровня освещенности

Датчик уровня освещенности (light sensor) определяет количество и интенсивность света в пространстве, окружающем устройство. Это позволяет приложениям настраивать изображение, например, уменьшая яркость, когда вокруг темно. Уменьшение яркости дисплея может увеличить срок работы устройства от одной зарядки батарей.

В листинге 4.9 показан пример чтения показателей датчика освещенности.

Листинг 4.9. Чтение показателей датчика освещенности

```
async private void ReadingChanged(object sender,
  LightSensorReadingChangedEventArgs e)
{
  await Dispatcher.RunAsync(CoreDispatcherPriority.Normal, () =>
{
  LightSensorReading reading = e.Reading;
  ScenarioOutput_LUX.Text = String.Format("{0,5:0.00}",
    reading.IlluminanceInLux);
  });
}
```

Подробнее о работе с датчиком освещенности вы можете узнать на странице http://msdn.microsoft.com/ru-ru/library/windows/apps/windows.devices.sensors.lightsensor.aspx.

Датчик положения в пространстве

Датчик положения в пространстве (orientation sensor) предоставляет матрицу, которая описывает положение устройства в пространстве, и кватернион (quaternion), помогающий корректировать точку зрения пользователя в некоторых приложениях, преимущественно игровых. В отличие от простого датчика ориентации, используемого при повороте изображения на экране, полнофункциональный датчик положения в пространстве обычно задействуют в играх для вывода изображения с учетом ориентации планшетного компьютера в пространстве. Кватернионы служат для описания ориентации и вращения объектов в пространстве.

В листинге 4.10 показано чтение информации с датчика положения в пространстве.

Листинг 4.10. Чтение показаний датчика положения в пространстве

```
async private void ReadingChanged(object sender,
  OrientationSensorReadingChangedEventArgs e)
{
  await Dispatcher.RunAsync(CoreDispatcherPriority.Normal, () =>
  {
    OrientationSensorReading reading = e.Reading;
    // Данные кватерниона
    SensorQuaternion quaternion = reading.Quaternion;
    ScenarioOutput_X.Text = String.Format("{0,8:0.00000}",
      quaternion.X);
```

```
    ScenarioOutput_Y.Text = String.Format("{0,8:0.00000}",
      quaternion.Y);
    ScenarioOutput_Z.Text = String.Format("{0,8:0.00000}",
      quaternion.Z);
    ScenarioOutput_W.Text = String.Format("{0,8:0.00000}",
      quaternion.W);
    // Данные матрицы поворота
    SensorRotationMatrix rotationMatrix = reading.RotationMatrix;
    ScenarioOutput_M11.Text = String.Format("{0,8:0.00000}",
      rotationMatrix.M11);
    ScenarioOutput_M12.Text = String.Format("{0,8:0.00000}",
      rotationMatrix.M12);
    ScenarioOutput_M13.Text = String.Format("{0,8:0.00000}",
      rotationMatrix.M13);
    ScenarioOutput_M21.Text = String.Format("{0,8:0.00000}",
      rotationMatrix.M21);
    ScenarioOutput_M22.Text = String.Format("{0,8:0.00000}",
      rotationMatrix.M22);
    ScenarioOutput_M23.Text = String.Format("{0,8:0.00000}",
      rotationMatrix.M23);
    ScenarioOutput_M31.Text = String.Format("{0,8:0.00000}",
      rotationMatrix.M31);
    ScenarioOutput_M32.Text = String.Format("{0,8:0.00000}",
      rotationMatrix.M32);
    ScenarioOutput_M33.Text = String.Format("{0,8:0.00000}",
      rotationMatrix.M33);
  });
}
```

Подробнее о датчике положения в пространстве вы можете узнать на странице http://msdn.microsoft.com/ru-ru/library/windows/apps/windows.devices.sensors.orientationsensor.aspx.

Выводы

Эта глава посвящена макетам приложений для Windows 8 и особенностям их поведения. Вы узнали о диспетчере визуальных состояний и о его роли в отделении логики уровня представления от программного кода приложения. Вы изучили возможности имитатора, которые позволяют при отладке приложения эмулировать такие возможности, как изменение географического расположения устройства и сенсорное взаимодействие с приложением. Кроме того, вы узнали, как работать с различными ориентациями дисплея и режимами просмотра приложения. Здесь состоялось ваше

знакомство с элементом управления `SemanticZoom`, который позволяет организовать удобное перемещение по большим наборам данных.

Также вы узнали о различных особенностях реализации сенсорного взаимодействия с приложениями в среде Windows 8. В рассмотренном примере было показано, как использовать мышь, клавиатуру и сенсорный дисплей для вызова различных команд и как обрабатывать события манипуляций. Вы узнали, как снабдить приложение экраном-заставкой и плитками, как, используя элемент управления `AppBar`, организовать размещение команд приложения, научились создавать страницу информации о программе с использованием возможностей чудо-кнопки Settings (параметры).

Устройства, работающие под управлением Windows 8, могут обладать различными датчиками, которые предоставляют сведения об ориентации устройства, освещенности, смещении, географическом положении и пр. Для работы с ними существуют специализированные прикладные программные интерфейсы, с помощью которых можно узнать, присутствует ли тот или иной датчик в системе, и прочесть показания нужного датчика. Датчики позволяют задать реакцию приложения на изменения, которые касаются физических параметров устройства, кроме того, они предоставляют пользователю сведения, касающиеся его окружения.

Среда разработки и шаблоны проектов позволили нам создать в качестве примера довольно сложное приложение, которое выводит на экран сгруппированные наборы элементов, поддерживает средства навигации, реализует операции добавления и удаления элементов. В обычных условиях, когда вы переходите в другое приложение, не закрывая предыдущее, его текущее состояние сохраняется в оперативной памяти. Благодаря этому, когда вы возвращаетесь в приложение, все в нем выглядит точно так же, как было в тот момент, когда вы переключились на другое. Однако нет никаких гарантий, что сведения о состоянии приложения все время будут храниться в оперативной памяти, а исполняющая среда Windows не завершит работу приложения.

Пользователи ожидают, что запуская закрытое ранее приложение, они увидят его в том же состоянии, в котором оно пребывало до закрытия. В следующей главе рассказывается о жизненном цикле приложения, а также о том, как узнать, когда работа приложения приостанавливается, после чего оно либо находится в памяти, либо полностью завершается, а при следующем обращении к нему запускается заново. Кроме того, вы узнаете, как приложения хранят сведения о состоянии. Вы можете восстановить состояние приложения при его повторном запуске, в результате у пользователя создастся приятное ощущение того, что приложение постоянно запущено.

Жизненный цикл приложения

5

В традиционных операционных системах семейства Windows временем жизни приложения управляет пользователь. Он запускает приложение, и приложение работает до тех пор, пока он же не решит его закрыть. В такой модели взаимодействия пользователя и приложения возникает одна проблема. Она заключается в том, что приложение продолжает потреблять системные ресурсы, в том числе ресурсы центрального процессора и оперативную память, даже тогда, когда пользователь с ним не работает. Это снижает производительность тех приложений, с которыми пользователь продолжает работать, а также приводит к быстрому истощению заряда батарей, если речь идет о портативном устройстве, отключенном от электросети.

Все это можно продемонстрировать на примере настольного режима Windows 8. Сейчас нам понадобится видеофайл, например одна из записей с конференции Microsoft, на которой была представлена платформа Windows 8. Такую запись можно найти на странице http://channel9.msdn.com/Events/BUILD/BUILD2011. (Обратившись к этому ресурсу, вы наверняка найдете там и другие видеоматериалы с самых последних мероприятий; их вы тоже можете загрузить.)

Выберите видеофайл и воспользуйтесь соответствующей ссылкой для загрузки видео либо среднего, либо высокого качества. Этот процесс может занять от нескольких минут до нескольких часов, в зависимости от размеров файла. Пока вы ожидаете завершения загрузки, запустите диспетчер

задач — инструмент, позволяющий просматривать список запущенных приложений. Диспетчер задач можно запустить, воспользовавшись одним из следующих способов:

- Введите поисковый запрос `task manager` на начальном экране Windows 8 и щелкните на значке программы, который появится в результатах поиска.
- Находясь на рабочем столе, щелкните правой кнопкой мыши на панели задач и выберите в появившемся меню команду Task Manager (Диспетчер задач).
- В любом месте системы воспользуйтесь комбинацией клавиш Ctrl+Shift++Esc.

При первом запуске диспетчер задач откроется в упрощенном режиме просмотра (рис. 5.1).

Рис. 5.1. Окно диспетчера задач в упрощенном режиме просмотра

В упрощенном режиме, в котором окно открывается по умолчанию, можно просмотреть список запущенных приложений. Для того чтобы увидеть более подробные сведения, щелкните на кнопке More details (Подробнее) в левой нижней части окна. В результате в окне диспетчера появятся дополнительные сведения, как показано на рис. 5.2. Здесь можно увидеть данные не только о запущенных приложениях и системных ресурсах, которые они используют, но и об исполняющихся фоновых процессах. Например, о драйверах, посредством которых операционная система взаимодействует с различными устройствами.

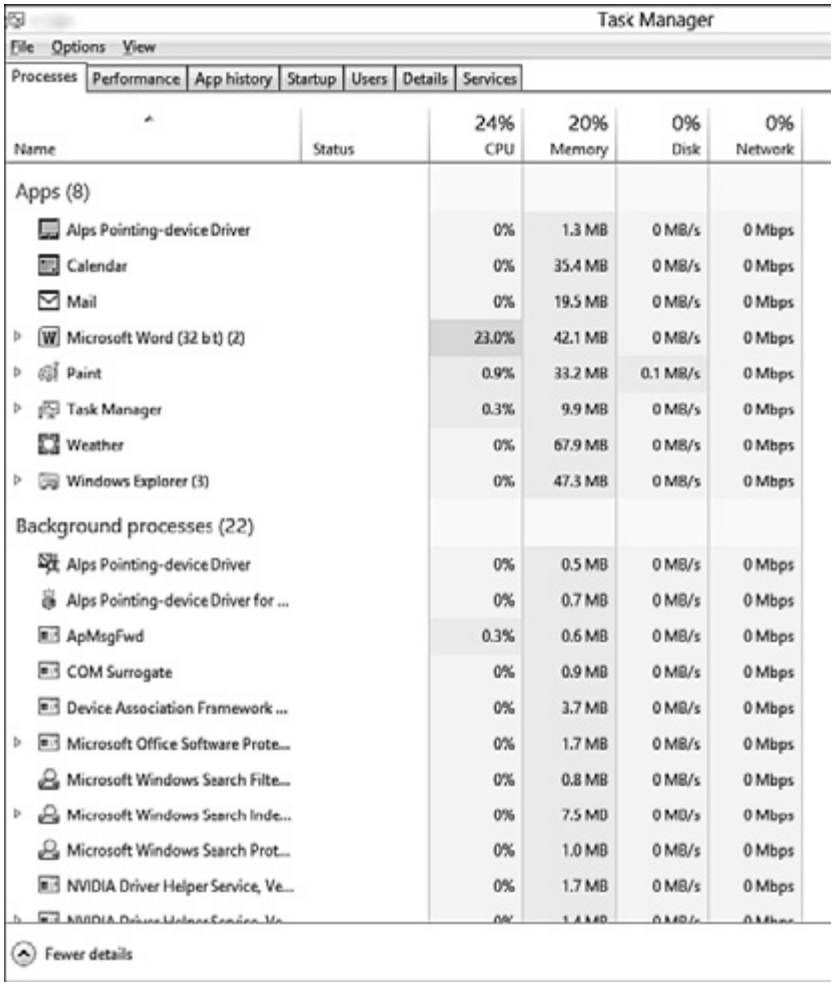


Рис. 5.2. Окно диспетчера задач в расширенном режиме просмотра

Убедитесь, что окно диспетчера задач открыто в расширенном режиме просмотра, и расположите его так, чтобы вы могли его видеть. Запустите загруженный видеофайл на воспроизведение. Для этого щелкните на значке файла в проводнике Windows правой кнопкой мыши и выберите в появившемся контекстном меню команду Play With Windows Media Player (Воспроизвести с помощью проигрывателя Windows Media). Вы увидите, что проигрыватель потребляет некоторое количество ресурсов процессора и памяти. Сколько именно — зависит от конфигурации вашей системы. Теперь запустите Блокнот (Notepad) и расположите его окно так, чтобы оно полностью перекрывало окно проигрывателя, где воспроизводится видео. При этом вы увидите, что потребление ресурсов проигрывателем не изменилось, хотя

видеоизображения на экране не видно. Приложение продолжает обрабатывать видеофайл даже тогда, когда вы, перекрыв его окно окном Блокнота, дали понять системе, что не собираетесь просматривать видео.

Приложения для Windows 8 функционируют иначе. Пользователь решает, какое приложение будет активным, и лишь это одно приложение работает (за исключением выполнения в режиме фиксации, когда активными могут быть два приложения, одно из которых занимает небольшую вертикальную часть экрана). А то, что должно происходить с приложениями, выполняющимися в фоновом режиме, система определяет самостоятельно. Если вы при создании приложения не учли этой особенности, такое приложение вряд ли понравится пользователям. Управление приложениями в среде выполнения Windows называется **управлением временем жизни процессов** (**Process Lifetime Management**, **PLM**).

Управление временем жизни процессов

Приложения для Windows 8 исполняются только тогда, когда они находятся на переднем плане, то есть являются активными. Это дает пользователю возможность сосредоточиться на основном приложении, с которым он взаимодействует. Приложения, которые располагаются на заднем плане (фоновые приложения), приостанавливаются. Это означает, что исполнение кода приложений «замораживается». Такие приложения не потребляют системные ресурсы и не загружают систему, расходуя энергию батарей. Тем не менее большую часть времени они остаются загруженными в оперативную память, что дает пользователю возможность быстро переключаться между приложениями. Поэтому когда пользователь возвращается к приложению, с которым он работал ранее, оно немедленно активизируется, порождая ощущение того, что оно все это время продолжало исполняться.

В определенных обстоятельствах приостановленное приложение может быть остановлено. Это зависит от имеющихся системных ресурсов. Например, если активному приложению нужен большой объем оперативной памяти, то при нехватке свободной памяти система может полностью остановить одно или несколько приостановленных приложений. Система анализирует объем оперативной памяти, занятой приостановленными приложениями, и останавливает в первую очередь те из них, которые потребляют больше памяти, чем другие.

Для того чтобы увидеть, как это работает, перейдите на начальный экран Windows 8, нажав клавишу Windows. Откройте диспетчер задач и расположите его поверх начального экрана. Для того чтобы это сделать, нужно раскрыть меню Options (Параметры) диспетчера задач и установить там флаг

Always on Top (Поверх остальных окон). Затем запустите какое-нибудь приложение, вернитесь на начальный экран и запустите другое приложение. Повторите это несколько раз. Вы заметите, что через несколько секунд приложения, запущенные ранее, но теперь не активные, приостанавливаются. Если в вашем окне диспетчера задач эти сведения о приложениях не выводятся, выберите команду View ▸ Status values ▸ Show suspended status (Вид ▸ Состояние ▸ Отображать приостановленные). На рис. 5.3 показан результат запуска нескольких приложений в ситуации, когда на переднем плане выполняется приложение Weather (Погода). Обратите внимание, что другие приложения для Windows 8 приостановлены — в столбце Status (Состояние) таких приложений указан вариант Suspended (Приостановлено).

Рис. 5.3. Приложения для Windows 8 в приостановленном состоянии

Если система сталкивается с нехваткой ресурсов, многие приостановленные приложения могут быть остановлены — в этом случае они исчезают из списка приложений диспетчера задач. Вы можете имитировать приостановку исполняемых приложений в режиме отладки в Visual Studio 2012. Как это сделать, вы узнаете в данной главе далее, но для начала важно понять особенности полного жизненного цикла приложений в рамках управления временем жизни процессов. Жизненный цикл приложения начинается с его *активизации* (activation).

Активизация приложения

Активизация — это первый этап жизненного цикла приложения. В предыдущих версиях Windows активизация обычно происходила после того, как пользователь запускал приложение: либо из меню Start (Пуск), либо из командной строки. В Windows 8 приложения всегда активизируются посредством контракта (о контрактах см. главу 2).

Наиболее распространенный контракт — это контракт запуска приложения (Launch). Этот контракт вызывается, когда вы касаетесь плитки приложения на начальном экране. Контракт запуска приложения активизирует приложение и вызывает метод `OnLaunched`. Еще одним контрактом, для обслуживания которого может быть активизировано приложение, является контракт общего доступа (Share). Он применяется в том случае, если речь идет о целевом приложении операции обмена данными, как в случае приложения ImageHelper (см. главу 2). Есть и другие контракты, использование которых может привести к активизации приложения. Приложение Windows8Application2 — это модифицированное учебное приложение из главы 4, в котором реализована поддержка контракта поиска (Search). Вы можете загрузить его с веб-страницы http://windows8applications.codeplex.com/.

Добавить поддержку этого контракта в приложение для Windows 8 довольно просто. В нашем учебном приложении это уже сделано, но если вы работаете над новым приложением, можете щелкнуть правой кнопкой мыши на проекте в проводнике решений (Solution Explorer) и выбрать в появившемся меню команду Add ▸ New Item (Добавить ▸ Создать элемент) либо воспользоваться комбинацией клавиш Ctrl+Shift+A. В появившемся окне создания новых элементов выберите вариант Search contract (Контракт поиска) и задайте имя для страницы результатов поиска, как показано на рис. 5.4.

Когда вы добавите новый элемент, автоматически будут выполнены некоторые действия. В дополнение к появлению в проекте новой страницы (объекта `Page`) в файле App.xaml.cs появится метод `OnSearchActivated`, предназначенный для обработки входящих поисковых запросов. Он предоставляет для события `OnLaunched` фрагмент кода, реализующий обработку запросов в запущенном приложении. Раздел объявлений манифеста приложения также обновится. В него будет добавлено объявление о поддержке приложением контракта поиска.

После того как все вспомогательные механизмы настроены, для реализации возможностей поиска можно использовать шаблоны. В примере результаты поиска выводятся на странице, которая содержит наборы фильтров для групп. Это позволяет пользователю выполнять поиск по всем элементам

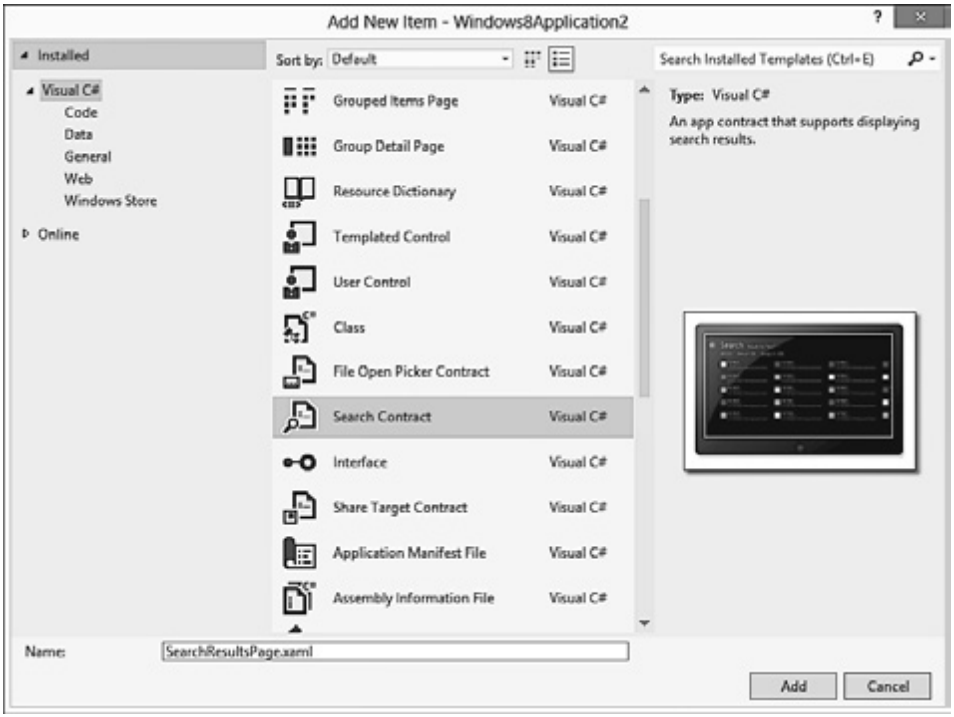


Рис. 5.4. Добавление в проект поддержки контракта поиска

или ограничиться конкретной группой элементов. Механизм поиска, кроме того, настроен на обработку входящих поисковых запросов вне зависимости от того, запущено приложение или нет.

Когда приложение активизируется для обслуживания любого контакта, оно выводит на экран заставку. Стандартная заставка представляет собой фоновый цвет и некое статическое изображение (о том, как задавать изображение и фоновый цвет для заставки, вы узнали в главе 4, а далее в этой главе узнаете о его настройке). Приложение оповещается об активизации, когда в файле **App.xaml.cs** вызывается соответствующий метод в зависимости от вида контракта. Для контракта запуска приложения (Launch) — это метод `OnLaunching`, для контракта поиска (Search) — метод `OnSearchActivated`, и т. д.

На то, чтобы вывести на экран свою стартовую страницу, у приложения есть 15 секунд. Если приложение не укладывается в этот временной диапазон, оно будет остановлено. Пользователь воспримет это как невозможность запуска приложения. Данное ограничение нужно учитывать при разработке приложений, которые нуждаются в загрузке больших объемов данных или длительной инициализации еще до вывода своей первой страницы.

В подобных ситуациях, чтобы выполнить все необходимые для запуска приложения действия, можно воспользоваться расширенной заставкой. Далее в этой главе мы поговорим о таких заставках.

Когда приложение активизируется, оно становится приложением переднего плана и исполняется до тех пор, пока пользователь не переключится на другое приложение, возможно, настольное. Когда пользователь переключается на другое приложение, приложение переднего плана уведомляется об этом событием `Suspending` (приостановка). Подобные уведомления означают, что приложение находится на грани приостановки, предлагая разработчику добавить в обработчик события код для сохранения состояния приложения перед его завершением.

Приостановка приложения

Обычный режим работы системы подразумевает приостановку приложения после того, как оно перестает быть приложением переднего плана. Для того чтобы упростить приостановку приложений в ходе их тестирования, Visual Studio 2012 предоставляет разработчику возможность самостоятельно инициировать этот процесс. Чтобы лучше познакомиться с механизмом приостановки, загрузите проект Windows8Application2.

Откройте файл App.xaml.cs и расположите точку останова в выделенной строке кода метода `OnSuspending`:

```
private static async void OnSuspending(object sender,
  SuspendingEventArgs e)
{
  var deferral = e.SuspendingOperation.GetDeferral();
  Debug.WriteLine(string.Format("{0} remaining",
  e.SuspendingOperation.Deadline - DateTime.Now));
  await SuspensionManager.SaveAsync();
  deferral.Complete();
}
```

Запустите приложение в режиме отладки. На панели инструментов вы должны увидеть значок, который похож на кнопку паузы с изображением приложения, — не путайте эту кнопку с кнопкой Pause (Пауза), которая относится к сеансу отладки в целом и просто останавливает исполнение программы. На рис. 5.5 вы можете видеть окно Visual Studio со всплывающей подсказкой к данной кнопке. Эту кнопку, которая имитирует приостановку приложения, вы можете использовать при отладке. Когда вы щелкнете на ней, то увидите, что сработала точка останова, так как был вызван метод

`OnSuspending` в объекте приложения. Изучите содержимое окна отладочной информации (если вы не видите этого окна, откройте его сочетанием клавиш Ctrl+Alt+O), после чего щелкните на кнопке Continue (Продолжить).

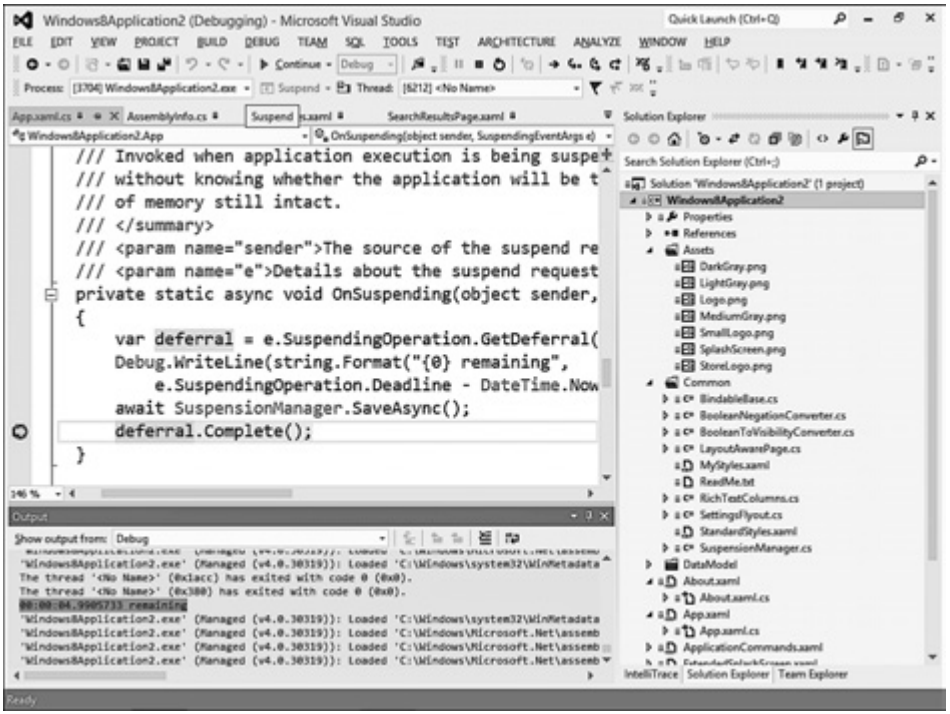


Рис. 5.5. Приостановка приложения в режиме отладки

Свойство `Deadline` содержит сведения о том, сколько времени есть у приложения для завершения операций, необходимых ему для приостановки. Если приложение в отведенное время не выполнит все, что необходимо, оно завершается. При выполнении асинхронных вызовов вы должны запросить отложенную операцию. С помощью аргументов, переданных событию, вы можете запросить отложенную операцию и затем исполнить набор асинхронных действий, таких как сохранение состояния приложения на диске. Наше приложение при обработке данного события сохраняет данные своего состояния, не выполняя каких-то других действий при приостановке. Когда приложение готово к приостановке, оно вызывает метод `Completed`, завершая подготовительные действия.

На данном этапе работы приложение все еще расположено в оперативной памяти, но не потребляет иных системных ресурсов. Если вы запустите диспетчер задач, то увидите, что уровень использования процессора составляет 0 %. То есть приложение не расходует заряд батарей, оно, фактически,

находится в режиме ожидания. Если вы возобновите его работу, либо активизировав приложение, либо воспользовавшись командой Resume (Возобновить), которая в Visual Studio 2012 расположена в том же меню, что и команда Suspend (Приостановить), приложение быстро без видимой задержки переключится в активное состояние.

СОВЕТ

К настоящему моменту вы уже знаете о том, как использовать отладчик Visual Studio 2012 для запуска приложений на локальном компьютере и с использованием имитатора. Есть и еще одна возможность: отладка на удаленном компьютере. Это особенно полезно, когда у вас есть планшетный компьютер, на котором вы можете отлаживать сенсорные возможности приложений, а разрабатывать их предпочитаете на настольном ПК или на ноутбуке. Для того чтобы удаленно отлаживать приложения, сначала нужно запустить на удаленном устройстве специальное приложение, настраивающее среду отладки. Затем вы можете выбрать удаленный компьютер в списке целевых платформ отладки. Прежде чем начнется процесс отладки, вам может понадобиться пройти процесс аутентификации. Полную инструкцию по настройке можно найти на странице http://msdn.microsoft.com/ru-ru/library/bt727f1t(v=vs.110).aspx.

Выполняя команды приостановки (Suspend) и возобновления (Resume), вы могли заметить еще одну команду: Suspend and shutdown (Приостановить и завершить работу). Эта команда служит для завершения приложения. Она имитирует ситуацию, когда приложение находится в приостановленном состоянии, а система при нехватке ресурсов завершает приложение для освобождения памяти.

Завершение приложения

Среда выполнения Windows завершает приложение, когда системе нужна дополнительная оперативная память, происходит переключение активного пользователя, выключают компьютер или имеет место фатальный сбой приложения. Не существует никакого события, уведомляющего о завершении приложения. Именно поэтому очень важно сохранять данные о состоянии приложения либо в ходе его работы, либо в обработчике события `OnSuspending`.

Обычно после запуска приложение выводит на экран свою домашнюю страницу независимо от того, было ли оно запущено впервые или пользователь запустил его снова после закрытия. Запускать приложение с применением сохраненного состояния следует лишь тогда, когда оно возобновляет

работу после завершения. К счастью, сведения о предыдущем состоянии всегда передаются в обработчик события `OnLaunched`. Это позволяет разработчику проверять предыдущее состояние и на основании этих сведений делать выводы о том, возвращается ли пользователь к завершенному приложению или повторно запускает закрытое им приложение.

Возобновление работы приложения

Приложение может зарегистрироваться для прослушивания события `Resuming`. Это событие вызывается лишь тогда, когда приложение приостановлено и находится в памяти в тот момент, когда пользователь снова его активизирует. В подобной ситуации не нужно восстанавливать состояние приложения, так как система сама поддерживает в готовности все потоки, стеки и хранилища данных приложения. Но если все так хорошо, зачем же тогда существует данное событие?

Благодаря этому событию приложения, которые работают с быстро устаревающими данными, могут обновить эти данные. Представьте себе погодное приложение, приостановленное, когда вы запустили игру, в которую затем играете несколько часов. Приложение остается в памяти, его потоки «заморожены». Затем вы возвращаетесь к нему. Что вы ожидаете увидеть? Без события `Resuming` вы получили бы устаревшие сведения, которые приложение выводило на экран перед приостановкой. Данное событие позволяет погодному приложению получить свежий прогноз погоды и вывести сведения о текущих погодных условиях, включая те или иные срочные сообщения.

Если приложение завершается, оно запускается заново, когда пользователь возвращается к нему. Событие `Resuming` в подобной ситуации не вызывается. Однако приложение может узнать о своем предыдущем состоянии, чтобы определить, что именно делать при запуске. Взгляните на метод `ExtendedSplashScreen_Loaded` в файле ExtendedSplashScreen.xaml.cs. Вы увидите там следующую проверку:

```
if (_activationArgs.PreviousExecutionState ==
  ApplicationExecutionState.Terminated
```

Если приложение было завершено, выполняется загрузка предыдущего состояния, и пользователь возвращается в то место приложения, в котором он был, когда переключился на другое приложение (как сохранять и восстанавливать состояния приложения, вы узнаете в этой главе далее). В противном случае приложение инициализируется и показывает пользователю свою домашнюю страницу. Другие возможные предыдущие со-

стояния приложения, проверку на которые можно применить при запуске: `ClosedByUser`, `NotRunning`, `Running` и `Suspended`.

На рис. 5.6 схематично показан жизненный цикл приложения.

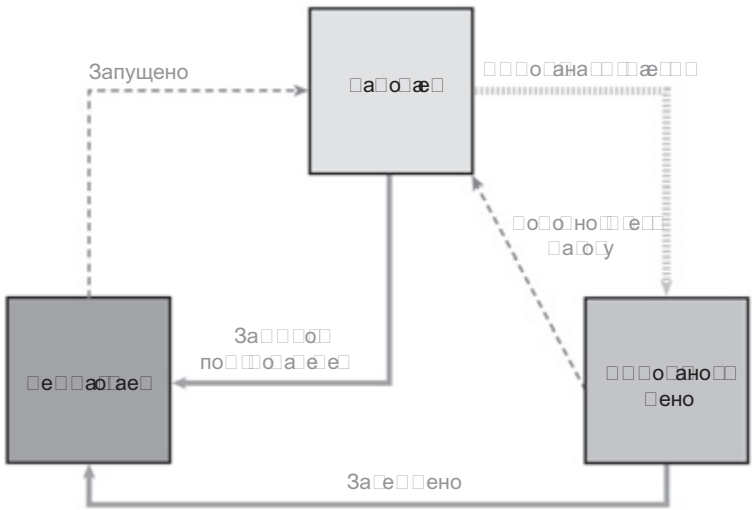


Рис. 5.6. Схема жизненного цикла приложения

Следующие рекомендации помогут вам создавать приложения, которые благодаря правильному управлению временем жизни процессов должны понравиться пользователю:

- **По мере выполнения приложения сохраняйте пользовательские данные.** Это предотвратит необходимость сохранять все данные единовременно при приостановке приложения. Так снижается риск того, что не удастся уложиться во время, отведенное системой для выполнения действий при приостановке.
- **Сохраняйте и восстанавливайте состояние приложения.** Важно обеспечить пользователю ощущение непрерывности работы с приложением, когда он возвращается к приложению.
- **Сохраняйте и восстанавливайте лишь метаданные сеанса (сведения о том, в каком месте приложения находится пользователь).** Другие данные следует обрабатывать в других местах приложения.

Теперь, когда вы знакомы с жизненным циклом приложения, вы можете использовать код шаблонов для управления его состоянием. Это поможет вам понять, как реализована навигация в приложениях для Windows 8. Когда приостановленное приложение запускают с помощью плитки на начальном экране, оно возобновляет работу, а не запускается заново.

Навигация

В приложениях для Windows 8 реализованы два основных варианта навигации: иерархическая и плоская. В иерархических приложениях пользователь сначала видит некое высокоуровневое представление данных и при необходимости спускается на более низкие уровни, содержащие детальные сведения. В нашем учебном приложении мы работали с группами коллекций данных, после чего спускались на уровень коллекций, и затем — на уровень отдельных элементов. Плоская модель навигации напоминает традиционные мастеры (wizards), которые позволяют пользователю перемещаться между отдельными этапами некоего процесса.

Когда вы запускаете почтовое приложение в первый раз, обычно все начинается именно с плоского сценария навигации. Мастер настройки приложения предлагает вам в пошаговом режиме ввести сведения о почтовом сервере, об учетных данных, проверить соединение, добавить введенные данные в список учетных записей. А вот в режиме обычной работы с самой почтовой программой применяется иерархический способ навигации. На верхнем уровне иерархии находятся учетные записи и сообщения, на следующем — тексты отдельных сообщений.

В приложениях для Windows 8, написанных с использованием языков XAML и C#, стандартным контейнером является элемент `Frame` (рамка). Взгляните на файл ExtendedSplashScreen.xaml.cs в проекте Windows8Application2. Программный код метода `ExtendedSplashScreen_Loaded` создает элемент `Frame`, оформляет подписку на событие и осуществляет переход на одну из страниц приложения. Прежде чем будет осуществлен переход на домашнюю страницу приложения, некоторые действия выполняются с помощью заставки (подробнее о нестандартных заставках мы поговорим далее в этой главе):

```
var rootFrame = new Frame();
rootFrame.Navigating += rootFrame_Navigating;
// здесь расположен дополнительный программный код
rootFrame.Navigate(target, parameter);
```

Объект `Frame` представляет собой визуальный контейнер, который поддерживает средства навигации. Единицей навигации в объекте `Frame` является объект `Page` (страница). Объект `Page` — это особый элемент управления, который разработан специально для целей навигации. Воспринимайте его как контейнер для визуальных элементов, которые вы хотите показать пользователю в различных местах приложения. Объект `Frame` позволяет осуществлять перемещение по страницам благодаря тому, что ему можно передавать типы объектов `Page`, соответствующие страницам, которые

нужно отобразить, и, возможно, параметры. В учебном приложении по умолчанию в корневой рамке приложения выводится главная страница с группами данных. Для этого используются данные о типе страницы и об имени коллекции групп из источника данных:

```
if (!rootFrame.Navigate(typeof(GroupedItemsPage), "ItemGroups"))
```

Вызов метода `Navigate` объекта `Frame` приводит к созданию экземпляра целевой страницы. Когда целевая страница загружена, вызывается метод `OnNavigatedTo`. Это позволяет разработчику задать пользовательский интерфейс. В учебном приложении этот метод служит для подключения данных к модели представления (подробности о моделях представления вы узнаете в главе 9) и для установки источника данных для отдельных элементов.

Объект типа `Frame` хранит журнал перемещений по страницам. Это позволяет ему работать как веб-браузеру, так как существует возможность перехода вперед и назад по истории навигации. Для выполнения этих действий применяются методы `GoBack` (перейти к предыдущей странице) и `GoForward` (перейти к следующей странице). Для того чтобы проверить, возможен ли переход вперед или назад, можно воспользоваться соответственно свойствами `CanGoBack` и `CanGoForward` объекта `Frame`.

Откройте файл GroupDetailPage.xaml. XAML-разметка для кнопки перехода назад выглядит следующим образом:

```
<Button x:Name="backButton" Click="GoBack"
  IsEnabled="{Binding Frame.CanGoBack,
  ElementName=pageRoot}"
  Style="{StaticResource BackButtonStyle}"/>
```

Кнопка доступна лишь в случае, когда в журнале навигации есть запись о ранее посещенной странице. Если она доступна и пользователь щелкает на ней, вызывается метод `GoBack`. Этот метод задан в базовом классе `LayoutAwarePage`, который является наследником объекта `Page`:

```
protected virtual void GoBack(object sender, RoutedEventArgs e)
{
  if (this.Frame != null && this.Frame.CanGoBack)
  this.Frame.GoBack();
}
```

Рисунок 5.7 иллюстрирует обычный рабочий процесс при возникновении навигационных событий. Объект `Frame` получает указание на переход к объекту `Page`, который называется *страница 2*. *Страница 1* получает сообщение

о происходящем благодаря событию `OnNavigatedFrom`. Если нужно предотвратить переход с некоторой страницы (например, если пользователь ввел данные в форму, но не сохранил их), *страница 1* может отменить переход, установив при обработке события флаг `Cancel`.

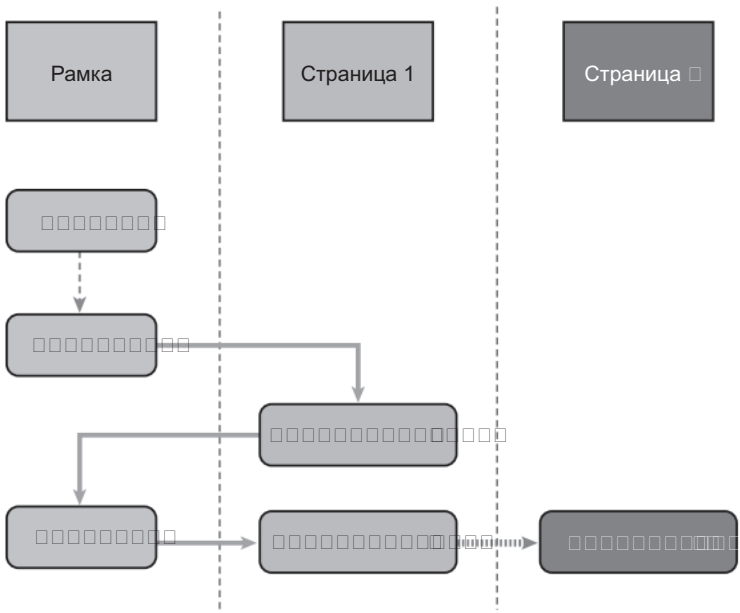


Рис. 5.7. Схема навигационных событий

Если *страница 1* не выполнила отмену перехода к другой странице, объект `Frame` выгружает *страницу 1* и вызывает событие `Navigated`. Это событие сообщает *странице 1*, что она была выгружена посредством метода `OnNavigatedFrom`. Затем экземпляр *страницы 2* размещается в видимой корневой рамке, данная страница получает сообщение об этом благодаря событию `OnNavigatedTo`. Объект `Frame` может либо создать новый экземпляр *страницы 2*, либо использовать уже имеющийся. Теперь информация о *странице 1* записана в журнал — так свойство `CanGoBack` сообщает о возможности перехода назад по истории навигации. В результате вызов метода `GoBack` для объекта `Frame` приводит к тому, что в рамке снова выводится *страница 1*.

Вы можете контролировать то, как объект `Frame` управляет хранением экземпляров страниц благодаря свойству `CacheSize` (размер кэша). Значение этого свойства представляет собой количество страниц, которые будут храниться в кэше. При вызове кэшированных страниц используются уже существующие экземпляры страниц, а не создаются новые. Для того чтобы управлять особенностями кэширования страниц, вы должны установить

свойство `NavigationCacheMode` объекта `Page` либо в значение `Enabled`, либо в значение `Required`. Когда используется значение `Required`, страница кэшируется независимо от значения параметра `CacheSize` объекта `Frame` и не учитывается при подсчете значения `CacheSize`.

Понимание особенностей работы подсистемы навигации упрощает управление положением пользователя при взаимодействием с приложением. Встроенные шаблоны Visual Studio 2012 предоставляют особый вспомогательный класс, который называется `SuspensionManager`. Он помогает сохранять и восстанавливать состояние приложения. Когда приложение загружается, корневой объект `Frame` регистрируется в объекте `SuspensionManager` файла App.xaml.cs:

```
SuspensionManager.RegisterFrame(rootFrame, "AppFrame");
```

Это позволяет объекту `SuspensionManager` наблюдать за перемещениями пользователя по приложению. Затем он сохраняет стек навигации и параметры, которые передаются страницам. В результате при возобновлении работы приложения он может восстановить как страницу, которую просматривал пользователь, так и историю навигации. Сведения о состоянии приложения сохраняются в локальном кэше с помощью API для работы с данными приложения.

API для работы с данными приложения

API для работы с данными приложения позволяет приложениям управлять собственными данными. Вы можете использовать этот прикладной программный интерфейс, чтобы сохранять данные о состоянии приложения, о заданных параметрах, локальный кэш и другие данные. Среда выполнения Windows поддерживает хранилище данных приложения. Доступ к хранилищу имеют лишь приложение и пользователь, для которых оно создано. Данные приложения сохраняются при обновлениях программы, однако после ее удаления из системы система их удаляет.

Приложение может работать с локальными данными трех типов. В табл. 5.1 приведены их характеристики и комментарии по особенностям их использования.

С каждым из хранилищ данных приложения можно взаимодействовать двумя способами. Первый способ заключается в работе с *параметрами приложения* (application settings). Он позволяет организовать хранение данных любых WinRT-типов. Данные можно хранить в виде отдельных или составных значений, а также в контейнерах, чтобы отделить наборы

данных друг от друга. Указанный способ организации данных предназначен для небольших объемов информации. В соответствии с документацией (см. страницу http://msdn.microsoft.com/ru-ru/library/windows/apps/hh464917.aspx) на самом деле параметры хранятся в реестре.

Таблица 5.1. Хранилища данных приложения

Вид данных	Описание
Local (Локальные данные)	Эти данные привязаны к приложению и пользователю на конкретном устройстве. Хранятся постоянно независимо от того, запущено приложение или нет и перезагружалась система или нет
Roaming (Перемещаемые данные)	Эти данные актуальны для приложения и пользователя на всех устройствах, на которых пользователь авторизовался посредством учетной записи Microsoft. Объем перемещаемых данных ограничен значением примерно 100 Кбайт
Temporary (Временные данные)	Данные этого вида могут быть удалены системой в любое время, они обычно служат для хранения локальных вспомогательных файлов или данных, которые не нуждаются в постоянном хранении в ситуации, когда приложение не запущено

Второй способ подразумевает работу с *файлами приложения* (application files). Это обычные файлы, которые можно записывать и считывать. В каждом хранилище выделяется место для файлов, выделенное конкретным приложению и пользователю. В хранилище можно создавать вложенные каталоги — вплоть до 32 уровней вложенности, и хранить столько файлов, сколько нужно. Файлы, записанные в хранилище временных данных, могут быть в любое время удалены. Файлы приложения, записанные в перемещаемое хранилище данных, будут синхронизированы с экземпляром приложения, установленного на другое устройство для пользователя с той же самой учетной записью Microsoft. На размер перемещаемых данных влияет ограничение, которое можно узнать, воспользовавшись свойством `ApplicationData.RoamingStorageQuota`.

Собственными локальными данными управляет приложение, поэтому обычно лишь оно само меняет их. Перемещаемые данные подразумевают иную схему работы, так как они могут измениться в ходе взаимодействия пользователя с другим экземпляром программы на другом устройстве. Когда данные приложения синхронизируются, поступление измененных перемещаемых данных вызывает событие `DataChanged`. Приложение может использовать это событие для того, чтобы отреагировать на изменения и обновить данные на локальном устройстве. Также об изменении данных

можно сообщить напрямую, воспользовавшись методом `SignalDataChanged` для вызова этого события.

В приложении Windows8Application2 локальные параметры служат для хранения сведений о текущей странице приложения, с которой работает пользователь. Для объекта `SuspensionManager` это вариант, предлагаемый по умолчанию. Причина, по которой применяются локальные, а не перемещаемые параметры, заключаются в том, что когда пользователь добавляет в программу новый элемент, он сохраняется лишь локально. Список элементов может быть довольно большим, поэтому его нет смысла хранить среди перемещаемых параметров приложения. Больше о данных приложения, в том числе и о том, как управлять большими наборами данных при работе с несколькими устройствами, вы узнаете в главе 6.

Хранилищем параметров приложения очень легко пользоваться. Если вам не нужно хранить историю навигации и параметры, передаваемые страницам, вы можете просто сохранить сведения о текущей странице, обновив свойство `NavigationPage` в файле App.xaml.cs:

```
set
{
  _navigatedPage = value;
  ApplicationData.Current.LocalSettings.Values["NavigatedPage"]
    = value.ToString();
}
```

Это действительно очень просто — вам достаточно установить значение, воспользовавшись выбранным ключом. Можно также сохранять данные сразу для некоторой группы элементов. Если вам нужно реализовать атомарную операцию, чтобы быть уверенным в том, что либо все значения группы будут сохранены, либо не будет сохранено ни одно из них, вы можете воспользоваться классом `ApplicationDataCompositeValue`. С его помощью можно подготовить атомарную группу значений, которую нужно сохранить в виде единого параметра. В файле App.xaml.cs сведения о текущей группе и элементе сохраняются следующим образом:

```
private void PersistCurrentSettings()
{
  var value = new ApplicationDataCompositeValue();
  value.Add("Group", _currentGroup == null ? string.Empty :
    _currentGroup.UniqueId);
  value.Add("Item", _item == null ? string.Empty : _item.UniqueId);
  ApplicationData.Current.LocalSettings.Values["Current"] = value;
}
```

Чтение значений параметров также не вызывает сложностей. Для того чтобы восстановить текущую страницу, приложение может просто прочитать значение строкового типа и затем конвертировать его в значение, тип которого нужен для перехода к странице:

```
target = Type.GetType(ApplicationData.Current.LocalSettings
  .Values["NavigatedPage"].ToString(), true);
```

При навигации будет осуществлен переход к текущему элементу или группе. Составные значения читают следующим образом:

```
var value = ApplicationData.Current.LocalSettings.Values["Current"]
  as ApplicationDataCompositeValue;
```

Затем можно получить конкретное значение и использовать его в запросе к внешней коллекции данных, чтобы получить из нее полные сведения об объекте:

```
var currentItemId = value["Item"];
CurrentItem = (from g in DataSource.ItemGroups
               from i in g.Items
               where i.UniqueId.Equals(currentItemId)
               select i).FirstOrDefault();
```

В запросе используется синтаксис запросов, интегрированных в язык (Language Integrated Query, LINQ). Подробнее о LINQ вы можете узнать на странице http://msdn.microsoft.com/ru-ru/library/bb397926(v=vs.110).aspx.

При необходимости параметры можно распределить по различным контейнерам. Для этого можно воспользоваться классом `ApplicationDataContainer`. Хранилище предоставляет метод `CreateContainer` для создания контейнеров, список текущих контейнеров можно получить через свойство `Containers`. Когда вы обращаетесь к конкретному контейнеру, вы можете получить доступ к словарю значений `Values` для записи и чтения параметров, которые хранятся в контейнере.

Хранилище не ограничивает разработчика использованием простой структуры данных вида ключ-значение. Здесь можно также создавать папки и файлы. Вместо того чтобы хранить сведения об отдельных страницах и параметрах так, как предложено ранее, можно задействовать стандартные вспомогательные механизмы объекта `SuspensionManager`, чтобы обрабатывать сведения о сценариях навигации, применяемых в вашем приложении. Объект `SuspensionManager` задает имя файла для хранения всех данных о состоянии приложения:

```
private const string sessionStateFilename = "_sessionState.xml";
```

Затем он использует класс DataContractSerializer для создания образа сохраняемого состояния навигации. Этот класс может представить в виде XML-данных все свойства и значения в экземпляре другого класса. Вот код, обеспечивающий выполнение этой операции для экземпляра XML-документа, который хранится в памяти:

```
MemoryStream sessionData = new MemoryStream();
DataContractSerializer serializer =
  new DataContractSerializer(typeof(Dictionary<string, object>),
    _knownTypes);
serializer.WriteObject(sessionData, _sessionState);
```

Далее выполняется запись полученных данных в хранилище. API данных приложения используется для создания файла, а затем сериализованный ранее поток записывается на диск:

```
StorageFile file = await ApplicationData.Current.LocalFolder
  .CreateFileAsync(sessionStateFilename,
    CreationCollisionOption.ReplaceExisting);
using (Stream fileStream = await file.OpenStreamForWriteAsync())
{
  sessionData.Seek(0, SeekOrigin.Begin);
  await sessionData.CopyToAsync(fileStream);
  await fileStream.FlushAsync();
}
```

Для того чтобы при возвращении к приложению восстановить ранее сохраненное состояние, выполняются обратные действия:

```
StorageFile file = await ApplicationData.Current.LocalFolder
  .GetFileAsync(sessionStateFilename);
using (IInputStream inStream = await file.OpenSequentialReadAsync())
{
  DataContractSerializer serializer = new DataContractSerializer(
    typeof(Dictionary<string, object>), _knownTypes);
  _sessionState = (Dictionary<string, object>)serializer
    .ReadObject(inStream.AsStreamForRead());
}
```

Скомпилируйте и запустите приложение, затем перейдите в группу, откройте страницу детального просмотра отдельного элемента. Завершите приложение, либо выполнив жест закрытия приложения, либо нажав клавиши Alt+F4. После этого откройте проводник и перейдите в следующую папку.

```
C:\users\{username}\AppData\Local\Packages\{packagename}
```

В этом пути замените *{username}* вашим именем пользователя, а *{packagename}* — именем пакета приложения из поля Package name (Имя пакета) вкладки Packaging (Упаковка) редактора манифеста приложения. На моем компьютере этот путь выглядит так:

```
C:\Users\Jeremy\AppData\Local\Packages\34F9460F-0BA2-4952-8627-
  ↳ 439D8DC45427_req6rhny9ggkj
```

Откройте папку LocalState, там должен присутствовать файл _sessionState.xml. В нем вы увидите сохраненные данные о состоянии приложения. Вот как выглядят XML-данные из нашего примера, содержащие сведения об истории навигации:

```
<KeyValueOfstringanyType>
<Key>Navigation</Key>
<Value i:type="a:string" xmlns:a="http://www.w3.org/2001/XMLSchema">
  1,3,2,33,Windows8Application2.GroupedItemsPage,
      ↳ 12,10,ItemGroups,32,
  Windows8Application2.GroupDetailPage,12,7,Group-2,31,
  Windows8Application2.ItemDetailPage,12,14,Group-2-Item-2
</Value>
</KeyValueOfstringanyType>
```

Больше о работе с данными вы узнаете в главе 6.

Подключены и активны

То, что фоновые приложения приостанавливаются и даже, возможно, завершаются, не означает, что они не могут оставаться на связи или казаться работающими. Один из эффективных способов поддерживать приложение в актуальном состоянии и обеспечивать пользователя свежими данными заключается в применении плиток и уведомлений (детали см. в главе 7).

Есть и другие случаи, когда приложение должно оставаться запущенным. Вот некоторые из них:

- **Воспроизведение аудиофайлов.** Определенные приложения, такие как проигрыватели для прослушивания музыки или подкастов, могут воспроизводить аудиофайлы в фоновом режиме. Для этого они используют возможности специальных элементов управления даже тогда, когда приложение не исполняется.

- **Загрузка файлов.** API фоновой передачи данных можно использовать для экономичной, в плане расходования электроэнергии, загрузки больших объемов данных.
- **Экран блокировки.** Специальные приложения экрана блокировки могут предоставлять пользователю некоторые сведения даже тогда, когда планшетный компьютер заблокирован. Например, почтовое приложение может сообщать о количестве непрочитанных почтовых сообщений.
- **Сокеты.** С помощью класса `ControlChannelTrigger` можно зарегистрировать приложение для обработки сетевых событий, уведомления о которых приложение получит, даже будучи приостановленным.

Ошибочно полагать, что фоновые приложения для Windows 8 полностью останавливаются. Комбинация служб, предлагаемых платформой, позволяет таким приложениям поддерживать связь с онлайновыми сервисами и предоставлять пользователю свежую информацию даже тогда, когда они не являются приложениями переднего плана. Применение таких прикладных программных интерфейсов позволяет приложениям экономно расходовать системные ресурсы и энергию батарей, интеллектуально реагируя на различные события.

Нестандартная заставка

Ранее в этой главе вы узнали, что приложение при запуске должно показать первую страницу в заданное время, иначе оно завершится. Когда вам нужно больше времени для начальной настройки приложения, вы можете во время инициализации приложения вывести на экран нестандартную заставку. В этом разделе вы узнаете, как ее настроить.

Приложение Windows8Application2 имитирует длительную загрузку приложения с помощью цикла. Этот цикл в режиме отладки выполняется медленно, так как в каждом его проходе данные выводятся в окно отладочной информации:

```
await Task.Run(() =>
                  {
                    for (var x = 0; x < 2000; x++)
                    {
                      Debug.WriteLine(x);
                    }
                  });
```

Цикл предназначен для имитации каких-либо операций, которые приложение выполняет при запуске. Обычно они включают в себя соединение с веб-сервисами для загрузки информации и чтение ранее сериализованных данных из кэша. Если для выполнения этих операций приложению нужно слишком много времени, Windows 8 остановит его через 15 секунд.

Для обхода этого ограничения предназначена нестандартная заставка, она обеспечивает предварительную загрузку данных до вывода основного макета приложения. В файле App.xaml.cs метод `OnLaunched` устанавливает экземпляр объекта `ExtendedSplashScreen` в качестве основного видимого объекта при запуске приложения и передает ему сведения об исходной заставке:

```
var splashScreen = args.SplashScreen;
var eSplash = new ExtendedSplashScreen(splashScreen, false, args);
splashScreen.Dismissed += eSplash.DismissedEventHandler;
Window.Current.Content = eSplash;
Window.Current.Activate();
```

Элемент управления `ExtendedSplashScreen` хранит ссылку на исходную заставку. Он содержит копию изображения обычной заставки и использует сведения о позиционировании заставки, предоставленные системой, для выравнивания этого изображения. Оно появится в том же месте, что позволяет незаметно перейти от исходной заставки к расширенной.

Когда элемент управления загружен, он просто вызывает метод `Initialize` для источника данных и в асинхронном режиме ожидает окончания загрузки элементов. Когда все необходимое загружено, элемент управления меняет видимый элемент на объект `Frame`, который служит для навигации, и осуществляет переход на первую страницу приложения. Код, реализующий эту логику, обычно находится в файле вспомогательного кода App.xaml.cs, но в данном случае он перемещен в элемент управления `ExtendedSplashScreen`, в итоге его исполнение может быть отложено до момента полной загрузки данных.

Поверх изображения, которое выводит заставка, размещается индикатор выполнения — элемент управления `ProgressRing`. Это позволяет оповестить пользователя о том, что приложение не зависло, а выполняет какие-то действия. Для того чтобы увидеть индикатор, запустите приложение в режиме отладки. Вы увидите изображение заставки, поверх которого отображается вращающийся индикатор. Он будет вращаться даже тогда, когда приложения приостановится, так как данная операция обрабатывается асинхронно и не блокирует UI-поток. Этот пример иллюстрирует очень простой механизм организации обратной связи. Приложения, выполняющие сложные процедуры инициализации, могут показывать пользователю и индикатор

выполнения, и текстовые сообщения, оповещающие о загрузке отдельных компонентов. Далее в этой книге мы рассмотрим более подробные примеры работы с этими механизмами, а здесь заставка просто исполняет некоторые простые действия, а затем загружает главную страницу приложения.

Выводы

В этой главе вы узнали о жизненном цикле приложения и о роли подсистемы управления временем жизни процессов (Process Lifetime Management, PLM). Приложения могут приостанавливаться и завершаться, но у них есть механизмы для сохранения и восстановления собственного состояния, данные которого можно синхронизировать между несколькими устройствами. Вы узнали о модели навигации на основе рамки, которую применяют приложения для Windows 8, и о том, как задействовать локальное хранилище данных для хранения сведений о странице приложения, просматриваемой пользователем. Локальное хранилище позволяет сохранять и считывать данные, которые привязаны к приложению и пользователю.

Также вы узнали о том, как создать нестандартную заставку, чтобы позволить приложению выполнить длительные операции инициализации. Без применения этого механизма приложение, превысившее заданное ограничение, принудительно завершается. В этой главе мы кратко рассмотрели вопросы работы с данными приложения на примере взаимодействия с локальной файловой системой. Приложения для Windows 8 могут работать с различными видами данных, получаемых как от веб-сервисов, так и из локальных файлов. Именно вопросам работы с данными посвящена следующая глава.

Данные

6

Данные играют важнейшую роль в большинстве приложений, поэтому очень важно понимать, как управлять данными, как преобразовать данные в информацию, с которой сможет работать пользователь. Приложения для Windows 8 могут взаимодействовать с данными различными способами. Вы можете сохранять локальные данные, получать транслируемый контент из Интернета, обрабатывать локальные ресурсы, хранящиеся в формате JSON. Вы также можете работать с XML-документами, задействовать WinRT-элементы управления, помогая пользователю выбирать файлы в файловой системе, и манипулировать коллекциями данных с применением языка структурированных запросов (Structured Query Language, SQL).

В этой главе вы узнаете о различных типах данных, с которыми могут работать приложения для Windows 8, о способах управления данными, об их загрузке, хранении и шифровании, о добавлении цифровой подписи и выполнении запросов к данным. Вы обнаружите, что WinRT предоставляет несколько удобных прикладных программных интерфейсов, которые серьезно упрощают работу с данными. В этой главе рассказывается как о самих этих интерфейсах, так и о том, как наилучшим образом интегрировать их в ваше приложение.

Параметры приложения

Ваше первое знакомство с параметрами приложения состоялось в главе 5. Вот типичные варианты использования параметров приложения:

- Доступ к простым параметрам можно получить посредством чудо-кнопки Settings (Параметры). Эти параметры могут быть синхронизированы между машинами (перемещаемые данные).

- В локальном хранилище данные сохраняются между сеансами работы приложения (локальные данные).
- Локальный постоянный кэш обеспечивает работоспособность приложения в тех случаях, когда оно внезапно теряет связь с сетевыми сервисами (локальные данные).
- Временные кэшированные данные используются для повышения производительности приложения (временные данные).

Значения параметров хранятся в простом словаре в виде базовых WinRT-типов данных. Можно хранить и более сложные типы. В главе 5 вы узнали, как самостоятельно сериализовать и десериализовать объект посредством записи в файл, расположенный в локальном хранилище данных. Вы выполняли сериализацию сложных типов, используя вспомогательный метод. Пример есть в классе `SuspensionManager`, который входит в шаблоны проектов. Вы можете найти файл SuspensionManager.cs на своем компьютере и изучить программный код.

Класс `SuspensionManager` использует объект `DataContractSerializer` для сериализации сложных типов в словаре:

```
DataContractSerializer serializer =
  new DataContractSerializer(typeof(Dictionary<string, object>),
    knownTypes_);
serializer.WriteObject(sessionData, sessionState_);
```

Сериализатор (в данном случае — класс `DataContractSerializerclass`) автоматически просматривает свойства целевого класса и формирует XML-данные, представляющие этот класс. XML-данные записываются в файл, который располагается в папке, выделенной для текущего приложения. Работа с локальными папками приложения, которые можно использовать для хранения других папок и файлов, организована аналогично работе с контейнерами параметров приложения (локальными, перемещаемыми, временными). Доступ к папке выполняется следующим образом:

```
StorageFile file =
  await
    ApplicationData.Current.LocalFolder. CreateFileAsync(filename,
  CreationCollisionOption.ReplaceExisting);
```

Доступ к папке перемещаемых и временных данных осуществляется похожим образом. Перечисление `CreationCollisionOption` позволяет генерировать имена файлов, которые не конфликтуют с существующими данными. Вот возможные значения этого перечисления (передаваемые методу создания файла):

- `FailIfExists`. Операция приведет к вбрасыванию исключения, если файл с таким именем уже существует.
- `GenerateUniqueName`. В ходе операции при необходимости к имени файла будет добавлена некоторая последовательность символов, позволяющая гарантированно создать уникальное имя файла.
- `OpenIfExists`. Если файл с заданным именем существует, вместо создания нового файла в ходе операции существующий файл открывается для записи.
- `ReplaceExisting`. Существующий файл, имя которого совпадает с заданным, будет перезаписан. В примере всегда осуществляется перезапись XML-файла с данными словаря.

После записи словаря вспомогательный механизм сериализации применяется для десериализации данных, когда приложение возобновляет работу после завершения:

```
DataContractSerializer serializer =
  new DataContractSerializer(typeof(Dictionary<string, object>),
    knownTypes_);
sessionState_ = (Dictionary<string, object>)serializer
  .ReadObject(inStream.AsStreamForRead());
```

Локальное хранилище данных пригодно не только для хранения данных о состоянии приложения. Как показано в главе 5, вы можете использовать его для хранения любых данных, например текстовых файлов и изображений. Обычно приложения разрабатывают с учетом применения локального хранилища в качестве локального кэша для хранения облачных данных. Такой подход позволит вашему приложению работать даже при отсутствии подключения к Интернету. Кроме того, в некоторых случаях, когда ожидается работа с сетевым соединением, отличающимся высокими задержками передачи данных, локальный кэш может повысить производительность приложения. В следующем разделе вы подробно узнаете о том, как работать с данными, используя среду выполнения Windows.

Доступ к данным и сохранение данных

Загрузите с сайта http://windows8applications.codeplex.com/ проект Wintellog, который входит в состав примеров к книге.

Для того чтобы работать с этим проектом, вам может понадобиться удалить TFS-привязки. Этот учебный проект демонстрирует несколько приемов

доступа к данным и их сохранения. Приложение загружает каналы блогов нескольких сотрудников Wintellect и кэширует их в составе локальных данных на устройстве, на котором оно запущено. Всякий раз, когда вы запускаете приложение, оно производит проверку на наличие свежих записей и загружает их.

Указанные блоги посвящены новинкам различных технологий, имеющих отношение к Windows 8, в частности Azure, SQL Server и другим. Вы можете знать авторов некоторых из них, например Джеффа Просиса (Jeff Prosise), Джеффри Рихтера (Jeffrey Richter) и Джона Роббинса (John Robbins).

В главе 5 вы узнали о различных хранилищах данных, а также о том, как их использовать для хранения параметров приложения или обычных файлов. Наше приложение с помощью параметров определяет, запускается ли оно в первый раз. Это занимает несколько минут, так как приложение загружает данные из блогов и обрабатывает веб-страницы для их вывода в собственном интерфейсе. Расширенная заставка требуется из-за того, что запуск приложения занимает некоторое время. Метод `ExtendedSplashScreen_Loaded` в файле SplashPage.xaml.cs проверяет, завершило ли приложение инициализацию:

```
ProgressText.Text = ApplicationData.Current.LocalSettings.Values
  .ContainsKey("Initialized") &&
   (bool)ApplicationData.Current. LocalSettings.Values["Initialized"]
  ? "Loading blogs..." :
"Initializing for first use: this may take several minutes...";
```

После завершения инициализации соответствующий флаг устанавливается в значение `true` (истина). Это позволяет приложению вывести оповещение о том, что при первом запуске ему может понадобиться дополнительное время на инициализацию. При последующих запусках для повышения производительности приложения основной объем данных будет загружаться из локального кэша:

```
ApplicationData.Current.LocalSettings.Values["Initialized"]
  = true;
```

В операциях по загрузке и сохранению данных участвуют несколько классов. Взгляните на класс `StorageUtility`. Он призван упростить сохранение объектов в локальном хранилище и чтение их оттуда при запуске приложения. В классе `SaveItem` вы можете увидеть код, обеспечивающий создание папки и файла с обработкой потенциально возможных коллизий, как описано в главе 5 (часть кода здесь для краткости не показана):

```
var folder = await
ApplicationData.Current.LocalFolder.CreateFolderAsync(folderName,
    CreationCollisionOption.OpenIfExists);
var file = await
    folder.CreateFileAsync(item.Id.GetHashCode().ToString(),
   CreationCollisionOption.ReplaceExisting);
```

Обратите внимание на то, что в имени метода есть ключевое слово `async`, а также на то, что файловым операциям предшествует ключевое слово `await`. Об этих ключевых словах мы поговорим в следующем разделе. В отличие от примера в главе 5, который предварительно настроен для записи данных определенного типа в хранилище, класс `StorageUtility` принимает параметры обобщенного типа, чтобы упростить сохранение данных любых сериализуемых типов. В этом коде задействован тот же самый механизм, который применяется для обработки данных сложных типов, передаваемых посредством веб-сервисов (подробнее о веб-сервисах мы поговорим далее в данной главе). Объект `DataContractJsonSerializer` служит для создания сериализованного аналога сохраняемого объекта:

```
var stream = await file.OpenAsync(FileAccessMode.ReadWrite);
using (var outStream = stream.GetOutputStreamAt(0))
{
  var serializer = new DataContractJsonSerializer(typeof(T));
  serializer.WriteObject(outStream.AsStreamForWrite(), item);
  await outStream.FlushAsync();
}
```

Файл создается посредством описанного вызова и используется для последующего получения потока. Экземпляру `DataContractJsonSerializer` передается тип сериализуемого класса. Сериализованный объект записывается в поток, соответствующий файлу, данные записываются на диск. Вся эта операция заключена в блок `try... catch`, чтобы организовать обработку ошибок, которые могут возникать в ходе работы с файловой системой. В операциях кэширования обычно применяется именно такой подход, так как если выполнить локальную операцию не удается, данные всегда можно повторно загрузить из облачного хранилища информации.

Для того чтобы увидеть, как работает сериализация и где хранятся файлы, запустите приложение, позвольте ему инициализироваться и вывести исходный сгруппированный список элементов. Нажмите сочетание клавиш Windows+R для открытия диалогового окна Run (Выполнить). Введите в этом диалоговом окне следующую строку:

```
%userprofile%\AppData\Local\Packages
```

Нажмите клавишу Enter, открыв папку данных приложения.

Это именно та папка, где хранятся данные приложения, соответствующие учетным данным пользователя, указанным при входе в систему. Для поиска нужной папки вы можете либо сравнить те имена папок, которые видите, с идентификатором пакета, либо ввести в поле поиска слово Groups, чтобы обнаружить папку, задействованную приложением Wintellog. Когда вы откроете папку локальных данных этого приложения, вы увидите несколько папок с именами в виде чисел, а также папку Groups, как показано на рис. 6.1.

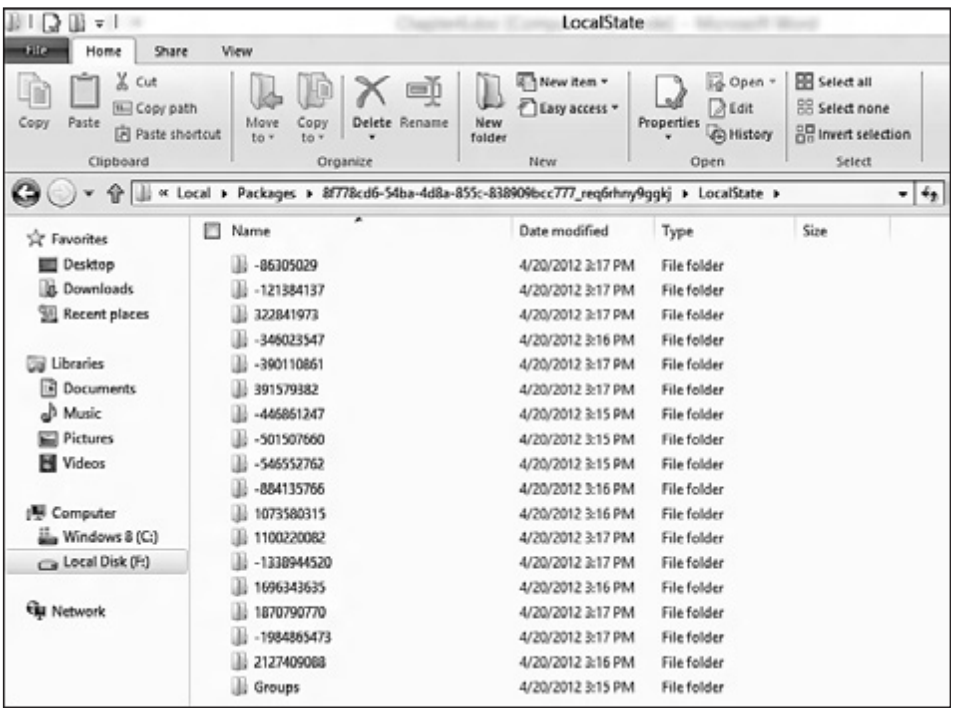


Рис. 6.1. Локальный кэш приложения Wintellog

Чтобы упростить именование объектов, приложение, назначая имена файлов, использует хэш-коды в качестве уникальных идентификаторов групп или элементов. Хэш-код — это числовое значение, которое упрощает сравнение сложных объектов. Детали вы можете узнать на странице http://msdn.microsoft.com/ru-ru/library/system.object.gethashcode.aspx.

Уникальность хэш-кодов не гарантируется, однако в случае с их строковыми представлениями весьма маловероятно, что комбинация из имени группы и сообщения приведет к коллизии. Папка Groups содержит список файлов для каждой группы. Откройте эту папку и просмотрите один из файлов

в Блокноте (Notepad). Вы увидите сериализованное значение объекта типа `BlogGroup` в формате JSON.

ОБОБЩЕННЫЕ ТИПЫ

Обобщенные типы — очень важный инструмент языка C#. Они позволяют создать кодовый шаблон, корректно работающий с различными типами, не будучи привязанным к какому-то конкретному типу. Метод SaveItem<T> — это шаблон для сохранения объектов неизвестного типа. Когда осуществляется его вызов из класса BlogDataSource, компилятор проверяет переданный ему тип и генерирует код, рассчитанный именно на этот тип данных. Здесь применяется открытое описание обобщенного типа, а при вызове осуществляется его закрытие с использованием конкретного типа данных. Возможно, вы знакомы с обобщенными коллекциями наподобие List<T>, но их можно использовать в гораздо более мощных и гибких решениях. Подробнее об обобщенных типах вы можете узнать на странице http://bit.ly/csharpgenerics.

JSON-файлы хранятся на диске в компактном формате. Вот как выглядит отформатированный для простоты фрагмент JSON-данных со сведениями о моем блоге:

```
{
  "Id" : "http://www.wintellect.com/CS/blogs/jlikness/default.
aspx",
  "PageUri" :
         "http://www.wintellect.com/CS/blogs/jlikness/default.
aspx",
  "Title" : "Jeremy Likness' Blog",
  "RssUri" : "http://www.wintellect.com/CS/blogs/jlikness/rss.aspx"
}
```

Здесь применяется весьма простой синтаксис. В фигурные скобки заключено описание объекта, выполненное с помощью списка ключей (имен свойств) и значений (тех, которые должны принимать соответствующие свойства). Если вы просмотрите любую сериализованную запись блога (содержащуюся в папке с именем, соответствующим хэш-коду группы), то обнаружите, что в свойстве `ImageUriList` для описания массива служат прямоугольные скобки:

```
"ImageUriList" : [
"http://www.wintellect.com/.../Screen_thumb_42317207.png",
"http://www.wintellect.com/.../someotherimage.png" ]
```

JSON

JSON (JavaScript Object Notation — нотация JavaScript-объектов) — это открытый стандарт хранения данных на основе текста. В случае веб-сервисов механизм сериализации, применяемый по умолчанию, использует язык XML для сохранения данных в структурированном документе. JSON опирается на иной подход, который стал популярным благодаря тому, что данные, закодированные в этом формате, имеют меньший объем в сравнении с XML-документами, и человеку проще их читать. Согласно этому стандарту для описания объекта служит синтаксис языка JavaSctipt. В результате для создания объекта достаточно, чтобы код обработал JavaScript-интерпретатор. Вы можете выполнить поиск в Интернете по ключевым словам «JSON Vizualizer», чтобы найти веб-сайты, поддерживающие JSON-код и позволяющие увидеть визуальную интерпретацию представляемого этим кодом объекта. Описание стандарта можно найти на странице http://json.org/.

Возможно, вы уже взглянули на класс `BlogGroup` и увидели, что сохранены не все его свойства. Например, количество элементов вычисляется при их загрузке в группу, поэтому данное свойство в сериализации не нуждается. Подобный прием требует, чтобы класс был маркирован с помощью атрибута `DataContract`, после чего нужно явным образом указать сериализуемые свойства. В случае с классом `BlogGroup` это выглядит так:

```
[DataContract]
public class BlogGroup : BaseItem
```

Сериализуемые свойства помечают атрибутом `DataMember`:

```
[DataMember]
public Uri RssUri { get; set; }
```

Если раньше вы разрабатывали веб-сервисы с помощью архитектуры Windows Communication Foundation (WCF), то должны быть знакомы с подобным подходом к пометке классов. Хотя вы, возможно, и не догадываетесь, что его можно использовать для прямого управления сериализацией, минуя стек веб-сервиса. По умолчанию выходным форматом для `DataContractSerializer` является XML, поэтому если вы хотите задействовать формат JSON, не забудьте указать объект `DataContractJsonSerializer`.

Процесс восстановления объектов примерно такой же. Файл на этот раз открывается для чтения. Та же подсистема сериализации служит для создания экземпляра объекта необходимого типа из сериализованных данных:

```
var folder = await
   ApplicationData.Current.LocalFolder.GetFolderAsync(folderName);
var file = await folder.GetFileAsync(hashCode);
var inStream = await file.OpenSequentialReadAsync();
var serializer = new DataContractJsonSerializer(typeof(T));
var retVal = (T)serializer.ReadObject(inStream.AsStreamForRead());
```

СОВЕТ

Обычно код инициализации класса размещают в его конструкторе. При использовании механизмов сериализации, предоставляемых системой, конструктор класса не вызывается. Это имеет смысл, так как подразумевается, что соответствующий объект уже создан и сериализуется то его состояние, которое отражает результаты инициализации. Если вам нужно, чтобы при десериализации объекта выполнялся некоторый код, вы можете задать метод, вызываемый подсистемой десериализации, пометив его атрибутом OnDeserialized. В примере Wintellog подобный подход реализован в классе BlogItem. Это сделано для того, чтобы гарантировать регистрацию события вне зависимости от того, как создан класс, с помощью ключевого слова new или в ходе десериализации.

Запуская приложение, вы можете заметить, что загрузка данных с веб-ресурсов, сохранение данных и восстановление элементов из кэша требуют времени. В среде выполнения Windows любые процессы, выполнение которых занимает больше нескольких миллисекунд, выполняются асинхронно. Их вызов отличается от вызовов синхронных механизмов. Для того чтобы понять разницу между ними, нужно хорошо понимать концепцию *программных потоков* (threading).

Быстродействие и программные потоки

В двух словах, программные потоки позволяют организовать одновременное исполнение различных процессов. Одна из задач процессора вашего устройства заключается в управлении этими потоками. Если в устройстве имеется лишь один процессор, несколько потоков исполняются на нем по очереди. Если же в устройстве установлено несколько процессоров, потоки могут исполняться на разных процессорах одновременно.

Когда пользователь запускает приложение, система создает главный программный поток приложения, который отвечает за решение большинства задач, включая реакцию приложения на ввод данных пользователем или на рисование изображений на сенсорном экране. То, что этот поток

поддерживает пользовательский интерфейс, привело к тому, что его стали называть потоком пользовательского интерфейса, или UI-потоком. По умолчанию код, который вы пишете, исполняется именно в UI-потоке, если только вы не решите выполнить какие-либо действия в другом программном потоке.

Проблема выполнения синхронных вызовов из UI-потока заключается в том, что все остальные механизмы, зависящие от этого программного потока, не будут функционировать до тех пор, пока не завершится работа вызванных механизмов. Если исполнение этого кода занимает несколько секунд, то подсистемы, ответственные за обработку событий касаний экрана или за обновление окна приложения, в это время исполняться не будут. Другими словами, приложение зависнет, не реагируя на команды пользователя.

Среда WinRT создана так, что позволяет избежать подобной ситуации. Это достигается благодаря асинхронным вызовами любых методов, на исполнение которых может понадобиться более 50 миллисекунд. Вместо того чтобы исполняться синхронно, эти методы делают свое дело в отдельном программном потоке, при этом UI-поток не блокируется. А после их завершения они возвращают результаты. Когда используется новое ключевое слово `await`, результаты автоматически продвигаются в вызывающий поток, обычно это UI-поток. Обычная ошибка здесь заключается в попытке обновления пользовательского интерфейса без возвращения в UI-поток. Подобная операция вызовет исключение, называемое нарушением правил доступа к потоку (cross-thread access violation), так как лишь UI-поток может управлять этими ресурсами.

Управление асинхронными вызовами в традиционном языке C# не было особенно сложным, но получаемый в результате код было сложно читать и поддерживать. В листинге 6.1 приведен пример использования традиционной модели, основанной на событиях. Завтрак (`breakfast`), обед (`lunch`) и ужин (`dinner`) происходят асинхронно, но предыдущий прием пищи (`meal`) должен завершаться прежде, чем может начаться следующий. В модели, основанной на событиях, регистрируется обработчик, соответствующий приему пищи, в итоге при завершении приема пищи может быть установлен флаг, сигнализирующий об этом событии. Для запуска процессов используются методы, имена которых в соответствии с соглашением об именовании заканчиваются словом `Async`.

Даже этот пример достаточно сложен. Каждый шаг требует регистрации (подписки) на событие завершения операции, чтобы затем, когда операция будет выполнена, управление было передано другому методу. То, что исполнение процесса происходит в различных методах, означает, что доступ к локальным переменным методов теряется, в итоге любая информация

должна передаваться между последовательными вызовами. Именно поэтому многие приложения обладают усложненной структурой и их непросто поддерживать.

Листинг 6.1. Асинхронный прием пищи с использованием модели, основанной на событиях

```
public void EatMeals()
{
  var breakfast = new Breakfast();
  breakfast.MealCompleted += breakfast_MealCompleted;
  breakfast.BeginBreakfastAsync();
}
void breakfast_MealCompleted(object sender, EventArgs e)
{
  var lunch = new Lunch();
  lunch.MealCompleted += lunch_MealCompleted;
  lunch.BeginLunchAsync();
}
void lunch_MealCompleted(object sender, EventArgs e)
{
  var dinner = new Dinner();
  dinner.MealCompleted += dinner_MealCompleted;
  dinner.BeginDinnerAsync();
}
void dinner_MealCompleted(object sender, EventArgs e)
{
  // готово;
}
```

В .NET 4.0 была включена библиотека параллелизма по задачам (Task Parallel Library, TPL), чтобы упростить поддержку параллельного (одновременно исполняющегося) и асинхронного кода. С помощью TPL описанные акты «приема пищи» можно описать в виде отдельных задач и исполнить их следующим образом:

```
var breakfast = new Breakfast();
var lunch = new Lunch();
var dinner = new Dinner();
var t1 = Task.Run(() => breakfast.BeginBreakfast())
  .ContinueWith(breakfastResult =>
     lunch.BeginLunch(breakfastResult))
  .ContinueWith(lunchResult => dinner.BeginDinner(lunchResult));
```

Подобный подход позволяет серьезно упростить процесс, но программный код по-прежнему сложен для чтения и понимания, его все еще непросто поддерживать. Во многих прикладных программных интерфейсах среды исполнения Windows применяется асинхронная модель. Для того чтобы еще больше упростить разработку приложений, в которых имеют место вызовы асинхронных методов, в Visual Studio 2012 были добавлены два новых ключевых слова: `async` и `await`.

Особенности использования ключевых слов async и await

Ключевые слова `async` и `await` упрощают асинхронное программирование. Метод, который планируется вызывать в асинхронном режиме и который не блокирует пользовательский интерфейс, маркируется ключевым словом `async`. Внутри этого метода можно вызывать другие асинхронные методы для выполнения длительных операций. Методы, маркированные ключевым словом `async`, могут возвращать одно из трех возможных значений.

Все операции в среде WinRT, маркированные ключевым словом `async`, возвращают один из четырех интерфейсов. То, какой интерфейс реализован, зависит от того, возвращает ли асинхронный метод результат вызывающему методу и поддерживает ли вызывающий метод отслеживание хода выполнения задачи. Доступные интерфейсы перечислены в табл. 6.1.

Таблица 6.1. Интерфейсы, доступные операциям, маркированным ключевым словом async

	Сообщается о ходе выполнения операции	**О ходе выполнения операции не сообщается**
Результаты возвращаются	IAsyncOperationWithProgress	IAsyncOperation
Результаты не возвращаются	IAsyncActionWithProgress	IAsyncAction

C# предлагает несколько подходов, реализующих заключение вызовов в асинхронные методы и их определение. Методы, которые вызывают асинхронные операции, помечаются ключевым словом `async`. Подобные методы, которые возвращают значение типа `void`, обычно являются обработчиками событий. Для обработчиков событий требуется именно пустой

возвращаемый тип. Например, если по щелчку на кнопке вы должны запустить асинхронную задачу, сигнатура обработчика событий может выглядеть следующим образом:

```
private void button1_Click(object sender, RoutedEventArgs e)
{
  // выполнить что-либо
}
```

Для того чтобы у вас появилась возможность дождаться завершения асинхронного вызова без блокирования UI-потока, вы должны добавить в сигнатуру метода ключевое слово `async`, например, так:

```
private async void button1_Click(object sender, RoutedEventArgs e)
{
  // выполнить что-либо
  await DoSomethingAsynchronously();
}
```

Если к методу, в котором указано ключевое слово `await`, не добавить модификатор `async`, это приведет к ошибке компиляции. Помимо рассмотренных особенностей, касающихся обработчиков событий, вам может понадобиться создать длительную задачу, которая должна завершиться до выполнения некоторого кода и при этом не возвращать никаких значений. Для реализации подобных методов нужно возвратить значение типа `Task`. Этот тип описан в пространстве имен `System.Threading.Tasks`. Например:

```
public async Task LongRunningNoReturnValue()
{
  await TakesALongTime();
  return;
}
```

Обратите внимание, что компилятор обрабатывает подобные конструкции автоматически. В вашем методе вы просто возвращаете управление без передачи какого-либо значения. Компилятор распознает в методе длительную задачу (`Task`) и автоматически создает объект `Task`. В итоге возвращаемый тип — это именно тип `Task`, который становится конкретным возвращаемым типом. В листинге 6.2 показано, как создать простой метод вычисления факториала и заключить его в оболочку асинхронного вызова. Метод `DoFactorialExample` асинхронно вычисляет факториал для числа 5, а затем помещает результат в свойство `Text` в виде строки.

Листинг 6.2. Создание асинхронного метода, который возвращает результат

```
public long Factorial(int factor)
{
  long factorial = 1;
  for (int i = 1; i <= factor; i++)
  {
    factorial *= i;
  }

  return factorial;
}

public async Task<long> FactorialAsync(int factor)
{
  return await Task.Run(() => Factorial(factor));
}

public async void DoFactorialExample()
{
  var result = await FactorialAsync(5);
  Result = result.ToString();
}
```

Обратите внимание на то, как просто, используя существующий синхронный метод (`Factorial`), представить его в асинхронном виде (`FactorialAsync`) и затем для получения результата вызывать с помощью ключевого слова `await` (`DoFactorialExample`). Вызов `Task.Run` — это именно тот механизм, который создает новый программный поток. Распределение задач по потокам иллюстрирует рис. 6.2. Обратите внимание, что UI-поток способен выполнять свои функции во время вычисления факториала, а полученный результат может быть показан пользователю.

Рис. 6.2. Распределение задач по программным потокам

В приведенном примере используется библиотека TPL, так как она поддерживается в предыдущих версиях .NET Framework. Асинхронное выполнение задач можно также обеспечить с помощью WinRT-методов, таких как

`ThreadPool.RunAsync`. Больше об асинхронном программировании в среде выполнения Windows вы можете узнать в центре разработчиков на странице http://msdn.microsoft.com/ru-ru/library/windows/apps/hh464924.aspx. Краткое руководство по использованию оператора `await` можно найти на странице http://msdn.microsoft.com/ru-ru/library/windows/apps/hh452713.aspx.

Лямбда-выражения

Параметр, который передается методу `Task.Run`, называется *лямбда-выражением* (lambda expression). Лямбда-выражение — это обычная анонимная функция. Она начинается с сигнатуры функции (если метод `Run` принимает параметры, они должны быть заданы внутри круглых скобок), а заканчивается телом функции. Я предпочитаю называть специальную стрелку `=>`, которая обозначает передачу в метод некоторых данных, *указателем передачи* (gosinta). Возьмем выражение из ранее приведенного кода, которое передается в `Task.Run`:

```
()=>Factorial(factor)
```

Его можно прочитать так: «В вызов метода `Factorial` с параметром `factor` ничего не передается». Вы можете использовать лямбда-выражения для вызова методов, создаваемых, что называется, «на лету». В предыдущих примерах, касающихся приемов пищи, был задан специальный метод, который обрабатывал события завершения. Для той же цели можно также использовать лямбда-выражение, например:

```
breakfast.MealCompleted += (sender, eventArgs)
      =>
  {
    // выполнить какие-либо действия
  };
```

В данном случае лямбда-выражение можно прочитать так: «Параметры `sender` и `eventArgs` передаются набору инструкций, который выполняет какие-то действия». Параметры события доступны в теле лямбда-выражения, как и локальные переменные, заданные во внешних методах. Лямбда-выражения являются удобным средством создания делегатов.

Работая с лямбда-выражениями, нужно учитывать некоторые особенности. Несмотря на то что лямбда-выражение присваивается переменной, ссылаться на нее в коде нельзя, поэтому вы не можете отменить регистрацию обработчика события, заданного с помощью лямбда-выражения. Лямбда-выражения, которые ссылаются на переменные в пределах метода,

захватывают эти переменные. В итоге они могут существовать даже вне области видимости обычных переменных метода (это происходит потому, что лямбда-выражение может быть вызвано после завершения метода). То есть вам нужно учитывать подобный побочный эффект. Больше о лямбда-выражениях вы можете узнать на странице http://msdn.microsoft.com/ru-ru/library/bb397687.aspx.

Вспомогательные средства ввода-вывода

Классы `PathIO` и `FileIO` предоставляют вспомогательные методы для записи и чтения файлов. Класс `PathIO` позволяет выполнять файловые операции посредством передачи абсолютного пути к файлу. Для того чтобы создать текстовый файл и записать в него некие данные, достаточно одной строки кода:

```
await PathIO.WriteTextAsync("ms-appdata:///local/tmp.txt",
"Text.");
```

Префикс `ms-appdata` — это особое URI-значение, которое указывает на локальное хранилище данных приложения. Вы также можете получить доступ к внедренным в приложение данным с помощью префикса `ms-appx`. В учебном приложении начальный список блогов загружается из JSON-файла, который хранится в файле Assets/Blog.js. Код для доступа к этому списку расположен в классе `BlogDataSource` (в папке DataModel). Доступ к файлу и загрузка данных осуществляются с помощью одной строки кода:

```
var content = await PathIO
  .ReadTextAsync("ms-appx:///Assets/Blogs.js");
```

Класс `FileIO` способен выполнять похожие операции. Однако вместо того, чтобы принимать путь к файлу и автоматически открывать файл, он принимает параметр типа `IStorageFile`. Используйте вспомогательные методы класса `FileIO`, когда у вас уже есть ссылка на файл или когда вам нужно сделать с файлом что-то такое, для чего одного лишь пути к файлу недостаточно.

В табл. 6.2 приведен список доступных методов упомянутых классов. Методы класса `PathIO` принимают абсолютный путь к файлу, методы класса `FileIO` — объект типа `IStorageFile` (полученный с помощью API для работы с хранилищами данных).

Таблица 6.2. Вспомогательные методы для работы с файлами из классов PathIO и FileIO

Имя метода	Описание
AppendLinesAsync	Присоединяет строки текста к заданному файлу
AppendTextAsync	Присоединяет текст к заданному файлу
ReadBufferAsync	Читает контент из заданного файла и записывает его в буфер
ReadLinesAsync	Читает контент из заданного файла и записывает его в текстовые строки
ReadTextAsync	Читает контент из заданного файла и записывает его в единственную текстовую строку
WriteBufferAsync	Записывает данные из буфера в заданный файл
WriteBytesAsync	Записывает байтовый массив в заданный файл
WriteLinesAsync	Записывает текстовые строки в заданный файл
WriteTextAsync	Записывает текст в заданный файл

Пользуйтесь этими вспомогательными методами при решении ваших задач. Они помогут значительно упростить код ваших приложений.

Внедренные ресурсы

Есть несколько приемов внедрения данных в приложение и чтения их оттуда. Обычная причина, по которой данные внедряют в приложение, заключается в необходимости предоставить некие начальные данные для локальной базы данных или кэша, параметры конфигурирования и специальные файлы, наподобие лицензионных соглашений. Вы можете внедрить в приложение любые ресурсы, в том числе изображения и текстовые файлы. В приложения, с которыми мы уже работали, включены графические ресурсы.

Для того чтобы задать способ внедрения ресурса, щелкните правой кнопкой мыши на его имени в обозревателе решений (Solution Explorer) и в появившемся контекстном меню выберите команду Properties (Свойства) либо выделите элемент и нажмите сочетание клавиш Alt+Enter. На рис. 6.3 показан результат выбора файла Blogs.js в папке Assets и вызова для него окна свойств. Обратите внимание на свойства Build Actions (Действие при построении) и Copy to Output Directory (Копировать в выходной каталог).

Рис. 6.3. Свойства ресурса

Когда вы устанавливаете свойство Build Actions (Действие при построении) в значение Content (Содержание), ресурс копируется в папку внутри пакета приложения. В дополнение к хранилищам данных, о которых вы узнали в главе 5, у каждого приложения имеется пакет, который развертывается в системе и хранит локальные ресурсы. Среди таких ресурсов, например, могут быть изображения.

Найти место, где развернут пакет приложения, можно с помощью класса `Package`:

```
var package = Windows.ApplicationModel.Package.Current;
var installedLocation = package.InstalledLocation;
var loc = String.Format("Installed Location: {0}",
  installedLocation.Path);
```

Самый простой способ доступа к подобным файлам заключается в использовании префикса `ms-appx`. Откройте файл BlogDataSource.cs. Файл Blog.js загружается в методе `LoadLiveGroups`, при этом префикс указывает на место расположения пакета:

```
var content = await PathIO.ReadTextAsync(
  "ms-appx:///Assets/Blogs.js");
```

Ресурсы, кроме того, можно внедрять в исполняемые файлы приложения. Эти ресурсы не видны в файловой системе, но к ним можно обращаться из программного кода. Установите свойство Build Actions (Действие при построении) ресурса в значение Embedded Resource (Внедренный ресурс). Доступ к таким ресурсам реализовать чуть сложнее.

Для того чтобы прочитать контент внедренного ресурса, нужно получить доступ к сборке приложения. Сборки — это строительные блоки приложений. Один из способов доступа к сборке заключается в использовании сведений о классе, который вы определили:

```
var assembly = typeof(BlogDataSource).GetTypeInfo().Assembly;
```

Сборка — это тот объект, в который внедрены ресурсы. Как только у вас есть ссылка на сборку, вы можете методом `GetManifestResourceStream` получить для ресурса поток ввода-вывода. Имя ресурса будет располагаться в пространстве имен сборки. Таким образом, доступ к ресурсу, который находится в корневом разделе проекта в пространстве имен проекта `Wintellog`, можно получить следующим образом:

```
Wintellog.ResourceName
```

Аналогично, для того чтобы обратиться к файлу ReadMe.txt, который расположен в папке Common, можно воспользоваться следующей ссылкой:

```
Wintellog.Common.ReadMe.txt
```

Файлы, подобные этому, обычно не внедряют в состав сборок, его свойство было соответствующим образом модифицировано лишь в демонстрационных целях. После того как вы получили для ресурса поток ввода-вывода, можете использовать средство чтения потоков, чтобы получить контент ресурса. Когда у вас есть ссылка на сборку, вы можете возвратить контент ресурса так:

```
var stream = assembly.GetManifestResourceStream(txtFile);
var reader = new StreamReader(stream);
var result = await reader.ReadToEndAsync();
return result;
```

Обычно внедренные ресурсы используют для того, чтобы усложнить нежелательный доступ к данным, скрывая их в сборке. Обратите внимание, что этот подход не позволяет добиться полного сокрытия данных, так как кто угодно, обладающий соответствующими инструментами, может проверить сборку и найти содержащийся в ней контент. Это касается и внедренных ресурсов. Внедрение ресурсов посредством установки свойства Build Actions (Действие при построении) в значение Content (Содержание) не только упрощает приложению доступ к ресурсам, но и предоставляет еще одну полезную возможность. Если вам нужно работать с несколькими ресурсами, внедрение позволяет просматривать содержимое файловой системы из папки, в которую установлено приложение.

Коллекции

Коллекции — это основные структуры данных, предназначенные для управления данными приложений. Классы коллекций реализуют обычные

интерфейсы, которые предоставляют единообразные методы для получения данных из коллекций и управления этими данными. Коллекции часто привязывают к UI-элементам. В примере Wintellog коллекция блогов, содержащая некоторое количество групп, привязана к элементу управления `GridView`. Коллекция постов внутри канала блога отображается внутри группы.

Среда выполнения Windows содержит набор стандартных типов коллекций. С помощью CLR эти типы автоматически отображаются на типы .NET Framework. В коде вы не ссылаетесь на WinRT-типы напрямую, а работаете с их .NET-эквивалентами, а CLR автоматически выполняет необходимые преобразования. В табл. 6.3 перечислены WinRT-типы и их .NET-эквиваленты, а также приведены краткие описания и примеры классов, которые реализуют конкретный интерфейс.

Таблица 6.3. Типы коллекций в среде исполнения Windows и в .NET

WinRT	.NET Framework	Пример	Описание
IIterable<T>	IEnumerable<T>	Большинство типов коллекций	Предоставляет интерфейс, который поддерживает перебор коллекции
IIterator<T>	IEnumerator<T>	Предоставляется посредством типа коллекции	Интерфейс для перебора коллекций
IVector<T>	IList<T>	List<T>	Коллекция, отдельные элементы которой доступны по индексу
IVectorView<T>	IReadOnlyList<T>	ReadOnly Collection<T>	Версия индексированной коллекции, которую нельзя модифицировать
IMap<K,V>	IDictionary<K,V>	Dictionary<K,V>	Коллекция значений, доступных по ключу
IMapView<K,V>	IReadOnly Dictionary<K,V>	ReadOnly Dictionary<K,V>	Версия коллекции, содержащей пары ключ/значение, которую нельзя модифицировать

WinRT	.NET Framework	Пример	Описание
IBindableIterable	IEnumerable	Предоставляется посредством коллекций, которые не являются обобщенными	Поддерживает перебор необобщенных коллекций
IBindableVector	IList	Нестандартные классы, реализующие IList	Поддерживает любые необобщенные коллекции, доступ к элементам которых может быть осуществлен по индексу

Один важный список, который не имеет эквивалента в WinRT, — это `ObservableCollection<T>`. Особенность этого списка в том, что он работает с системой привязки данных, о которой вы узнали в главе 3. `ObservableCollection<T>` реализует интерфейс `INotifyCollectionChanged`, предназначенный для уведомления слушателей при изменении списка, например при добавлении или удалении элементов, а также при обновлении всего списка.

В целях обеспечения высокой производительности система привязки данных не осуществляет непрерывную проверку списков, привязанных к UI-элементам. Вместо этого первоначально привязанный список используется для создания элементов управления на экране. Когда вы работаете со списком, система привязки данных получает уведомления посредством события `CollectionChanged` и на основе перечня добавленных и удаленных элементов может обновлять выводимые на экран элементы управления. Без данного интерфейса единственная возможность обновлять пользовательский интерфейс при изменении в соответствующем списке заключается в вызове события `PropertyChanged` для свойства, которое предоставляет список. Такой подход неэффективен, так как приводит к обновлению всего списка вместо обновления лишь изменившихся позиций.

Технология LINQ

Одно из важнейших преимуществ коллекций заключается в возможности написания запросов к ним с использованием синтаксиса запросов, интегрированных в язык (Language Integrated Query, LINQ). Эта технология расширяет синтаксис языка C#, поскольку для запросов и обновлений данных предлагает готовые решения в виде паттернов. LINQ работает с провайдерами для различных типов хранилищ данных, таких как базы данных

(SQL) или XML-документы. LINQ-провайдер объектов поддерживает классы, которые реализуют интерфейс `IEnumerable`, в результате этот провайдер может работать с большинством коллекций.

LINQ-провайдер объектов реализован в виде набора методов расширения для существующего интерфейса `IEnumerable`. Эти методы расширения, определенные в пространстве имен `System.Linq`, позволяют добавлять методы к существующим типам без необходимости создания новых типов. Метод расширения — это особый вид статического метода, который использует специальный модификатор `this` для первого параметра. Больше о методах расширения вы можете узнать на странице http://msdn.microsoft.com/ru-ru/library/bb383977(v=vs.110).aspx/.

LINQ-запрос выполняется в три этапа. Первый подразумевает предоставление источника данных или коллекции, к которой планируется делать запрос. Второй этап заключается в задании самого запроса. Последний этап — это собственно выполнение запроса. Важно понимать, что само по себе создание запроса не ведет к выполнению каких-либо операций над источником данных. Запрос исполняется лишь тогда, когда это нужно разработчику, обработка результатов производится после их получения. Это называют *отложенным исполнением* (deferred execution).

Существует множество разновидностей LINQ-запросов. Также поддерживается разнообразный синтаксис запросов. В классе `BlogDataSource` проекта Wintellog есть метод, который называется `LinqExamples`. Этот метод никогда не вызывается, но вы можете использовать его, чтобы увидеть примеры синтаксиса LINQ-запросов и сами запросы. Первый вид синтаксиса называется *синтаксисом запросов* (query syntax). Он напоминает синтаксис T-SQL, который вы, возможно, применяли в работе с базами данных. Второй вид синтаксиса ориентирован на методы, его называют *синтаксисом методов* (method syntax). Эта разновидность синтаксиса ориентирована на использование лямбда-выражений.

Следующая группа примеров демонстрирует применение этих двух видов синтаксиса, начиная с синтаксиса запросов.

Запросы

Для обработки коллекций и получения из них нужных данных можно использовать простые запросы. Следующие примеры демонстрируют получение списка строк, которые представляют собой заголовки из коллекции, содержащей группы блогов:

```
var query = from g in GroupList select g.Title;
var query2 = GroupList.Select(g => g.Title);
```

Фильтрация

Фильтры позволяют вводить ограничения на данные, получаемые с помощью запроса. Организовывать фильтрацию можно с использованием обычных функций, которые сравнивают свойства и манипулируют ими. В следующих примерах список групп фильтруется для отбора только тех заголовков, которые начинаются с латинской буквы «A»:

```
var filter = from g in GroupList
  where g.Title.StartsWith("A")
  select g;
var filter2 = GroupList.Where(g => g.Title.StartsWith("A"));
```

Сортировка

Сортировать элементы можно в восходящем и нисходящем порядке, а также при необходимости выполнять сортировку по нескольким значениям свойств. Следующие запросы сортируют блоги по заголовкам:

```
var order = from g in GroupList
  orderby g.Title
  select g;
var order2 = GroupList.OrderBy(g => g.Title);
```

Группировка

Весьма мощная черта LINQ-запросов — возможность группировки сходных результатов. Она особенно полезна в приложениях для Windows 8 при реализации списков элементов управления, которые поддерживают группировку. Следующие запросы создают группы на основе первой буквы заголовка блога:

```
var group = from g in GroupList
  group g by g.Title.Substring(0, 1);
var group2 = GroupList.GroupBy(g =>
  g.Title.Substring(0, 1));
```

Объединения и проекции

Несколько источников данных можно объединить и спроецировать в виде нового типа, который содержит лишь необходимые свойства. Следующий синтаксис запроса объединяет элементы одного блога с элементами другого на основе даты отправки записи, а затем проецирует результаты в новый

класс, содержащий свойства для заголовка-источника (`source`) и целевого заголовка (`target`):

```
var items = from i in GroupList[0].Items
            join i2 in GroupList[1].Items
            on i.PostDate equals i2.PostDate
            select new
            { SourceTitle = i.Title, TargetTitle = i2.Title };
```

Вот тот же запрос, реализованный с помощью лямбда-выражений:

```
var items2 = GroupList[0].Items.Join(
    GroupList[1].Items,
    g1 => g1.PostDate,
    g2 => g2.PostDate,
     (g1, g2) => new { SourceTitle = g1.Title,
        TargetTitle = g2.Title });
```

В этом разделе мы рассмотрели лишь некоторые из возможностей LINQ-запросов. Больше об этой технологии вы можете узнать, прочтя статьи и учебные руководства, ссылки на которые можно найти на странице http://msdn.microsoft.com/ru-ru/library/bb383799(v=vs.110).aspx.

Веб-контент

Среда исполнения Windows упрощает загрузку и обработку веб-контента. Для того чтобы получить доступ к веб-страницам, можно воспользоваться классом `HttpClient`. Он похож на класс `WebClient`, с которым, вероятно, хорошо знакомы Silverlight-разработчики. Этот класс служит для отправки запросов и получения ответов серверов по протоколу HTTP. Его можно применять для отправки стандартных HTTP-запросов любых типов, в том числе `GET`, `PUT`, `POST` и `DELETE`. HTTP-клиент возвращает экземпляр `HttpResponseMessage` с кодом статуса ответа и заголовками. Если операция проходит успешно, его свойство `Content` предоставляет контент полученной веб-страницы.

Класс `BlogDataSource` содержит вспомогательный метод, который предоставляет экземпляр `HttpClient`. Метод задает размер буфера в расчете на загрузку больших страниц и предоставляет для запроса пользовательский агент (user agent). Пользовательские агенты обычно служат для идентификации браузера, который выполняет веб-запрос. В случае с программным доступом к сайтам вы можете передать сведения, которые предоставят серверу данные о приложении и ожидаемых параметрах совместимости.

Передача агента, совместимого с мобильными устройствами, может привести к тому, что веб-сервер возвратит страницу, оптимизированную для мобильного просмотра.

ДОСТУП К СЕТИ

Использование класса HttpClient требует объявления в манифесте приложения соответствующих возможностей. То, какие именно это будут возможности, зависит от расположения веб-сервера, на котором находятся нужные данные. Если веб-сервер расположен в домашней или частной сети, нужно включить режим Private Networks (Client & Server) (Частные сети (клиент и сервер)). В большинстве случаев доступ осуществляется к серверам, которые расположены в Интернете. Для реализации подобной функциональности нужно включить режим Internet (Client) (Интернет (клиент)), который при создании нового проекта активизируется по умолчанию.

Среда выполнения Windows упрощает асинхронный доступ к странице и обработку полученных данных. Две следующих строки кода вызывают веб-клиент и получают с его помощью страницу:

```
var client = GetClient();
var page = await client.GetStringAsync(item.PageUri);
```

Изображения не всегда внедрены в RSS-каналы, поэтому код получает страницу записи, а затем обрабатывает ее для загрузки необходимых изображений. Данная операция выполняется с помощью регулярных выражений. Синтаксис регулярных выражения удобно использовать для описания паттернов поиска определенных данных в текстах. Это делает данную технологию идеальной для решения таких задач, как выборка из целевых документов чего-то наподобие HTML-тегов.

РЕГУЛЯРНЫЕ ВЫРАЖЕНИЯ

Регулярные выражения предоставляют мощный синтаксис для поиска и замены паттернов в текстах. Первые средства обработки регулярных выражений были интегрированы в ранние дистрибутивы Unix как части текстовых редакторов и утилит командной строки, предназначенных для обработки больших объемов данных. Для обработки текста с использованием регулярных выражений в .NET Framework имеется объект System.Text.RegularExpressions.Regex. Большинство подобных операций требуют двух строк: целевая строка содержит текст для обработки, строка паттерна — само регулярное выражение. Для того чтобы больше узнать о синтаксисе регулярных выражений и об их использовании в .NET, обратитесь к ресурсу http://msdn.microsoft.com/ru-ru/library/hs600312.aspx.

Первое выражение обрабатывает все теги `image` в исходном коде веб-страницы:

```
public const string IMAGE_TAG = @"<(img)\b[^>]*>";
private static readonly Regex Tags = new Regex(IMAGE_TAG,
  RegexOptions.IgnoreCase | RegexOptions.Multiline);
var matches = Tags.Matches(content);
```

Затем обрабатывается каждый найденный тег, чтобы получить адрес изображения из его атрибута `src`. Полученные адреса изображений используются для создания экземпляров `Uri`, которые записываются в свойство `ImageUriList` объекта, представляющего запись блога. Это свойство реализовано в виде коллекции `ObservableCollection`, что позволяет организовать отправку уведомлений при добавлении новых изображений. Для каждой записи выводится случайно отобранное изображение. Изображение находится в Интернете, но если ранее оно уже загружалось, Windows 8 задействует кэшированную копию изображения в случае, если компьютер пользователя временно не подключен к Интернету.

Транслируемый контент

Транслируемый контент — это информация сайтов, доступная другим сайтам посредством специальных каналов. Обычно эти каналы представлены в формате XML с использованием либо технологии RSS, либо Atom. RSS — это аббревиатура от RDF Site Summary (сводка RDF-данных сайта), хотя обычно ее расшифровывают как Real Simple Syndication (очень простая трансляция). В свою очередь, аббревиатура RDF расшифровывается как Resource Description Framework (схема описания ресурсов). Оба формата требуют применения стандарта XML к блогам, веб-сайтам и другим поставщикам контента с целью единообразного представления данных, чтобы другие программы могли загружать и использовать эти данные.

Спецификацию RSS можно найти по адресу http://www.rssboard.org/rss-specification.

Описание протокола публикации данных Atom доступно по адресу http://atompub.org/.

Оба формата предусматривают формирование *канала* (feed), который представляет собой набор однородных записей (тем, статей, записей блога). Каждая запись может содержать сведения о дате публикации, об авторе, набор ссылок, которые ведут к ее исходному ресурсу, и мультимедийный контент, такой как изображения и видеоклипы. Для того чтобы работать с этими данными, вы можете либо ознакомиться с документацией

и написать собственную XML-подсистему, либо найти подсистему синтаксического разбора от стороннего производителя.

Среда выполнения Windows предоставляет класс `SyndicationClient`, который упрощает взаимодействие с каналами. Этот класс из пространства имен `Windows.Web.Syndication` можно применять для асинхронной загрузки каналов, он может использовать переданные ему учетные данные для подключения к ресурсам, требующим авторизации. При передаче ему URL-адреса ресурса он может обрабатывать каналы в формате Atom (версий 0.3 и 1.0) и RSS (версий 0.91, 0.92, 1.0 и 2.0), представляя контент в виде общей объектной модели (COM).

Учебное приложение загружает данные канала с помощью всего четырех строк кода. Без двух из них можно обойтись, они служат лишь для того, чтобы при выполнения запроса к веб-сайту задействовать преимущества кэша браузера и указать нестандартный пользовательский агент. Вспомогательный метод `GetSyndicationClient` возвращает клиент, который в классе `BlogDataSource` имеет несколько свойств, заданных по умолчанию:

```
private static SyndicationClient GetSyndicationClient()
{
  var client = new SyndicationClient
                  { BypassCacheOnRetrieve = false };
  client.SetRequestHeader("user-agent", USER_AGENT);
  return client;
}
```

Для использования клиента достаточно вызывать метод получения канала, передав ему адрес этого канала:

```
var client = GetSyndicationClient();
var feed = await client.RetrieveFeedAsync(group.RssUri)
```

Если операция произведена успешно, ее результатом становится объект типа `SyndicationFeed`. Этот экземпляр содержит данные о расположении канала, категории или теги, которые имеются в канале, сведения об участниках канала и, конечно, о позициях, отправленных в канал. Каждый объект `SyndicationItem` в канале хранит сведения о расположении позиции, о категориях или тегах, относящихся к этой позиции, заголовок и конент позиции, а также, возможно, краткое описание.

Вы можете просмотреть код примера, чтобы увидеть, как просто с помощью описанных механизмов можно обрабатывать каналы и получать необходимые данные. При этом не нужно указывать формат канала, так как класс автоматически получает сведения о его формате, анализируя контент. Трансляция — это мощная технология предоставления контента и работы с ним в приложениях для Windows 8.

Потоки ввода-вывода, буферы и байтовые массивы

Для чтения данных из файлов, веб-сайтов или других ресурсов в .NET обычно применяют потоки ввода-вывода. Поток ввода-вывода позволяет передать данные в структуру, которую можно читать и которой можно управлять. Потоки ввода-вывода могут также предоставлять возможность передачи контента из структуры данных обратно в поток для последующей записи этих данных. Некоторые потоки ввода-вывода поддерживают операции поиска. Поиск в потоке ввода-вывода очень похож на перемещение по сценам в DVD-фильме.

Потоки ввода-вывода обычно записывают в байтовые массивы. Байтовый массив — это предпочтительный механизм для работы с двоичными данными в .NET. Его можно использовать для манипуляций данными, такими как содержимое файла или пикселы, формирующие изображение. Множество классов, реализующих потоки ввода-вывода в .NET, поддерживает преобразование байтового массива в поток ввода-вывода или чтение потока ввода-вывода в байтовый массив. Также вы можете конвертировать данные других типов в байтовый массив с помощью класса `BitConverter`. Следующий пример иллюстрирует преобразование 64-битного целого числа в массив из 8 байт (8 байт × 8 бит = 64 бит) и последующее обратное преобразование:

```
var bigNumber = 4523452345234523455L;
var bytes = BitConverter.GetBytes(bigNumber);
var copyOfBigNumber = BitConverter.ToInt64(bytes, 0);
Debug.Assert(bigNumber == copyOfBigNumber);
```

Среда выполнения Windows представляет интерфейс `IBuffer`, который служит чем-то вроде посредника между байтовым массивом и потоком ввода-вывода. Интерфейс содержит два члена: свойство `Capacity` (максимальное количество байтов, которое может хранить буфер), и свойство `Length` (текущее количество байтов в буфере). Многие операции в WinRT либо принимают, либо возвращают экземпляр `IBuffer`.

Преобразование потоков ввода-вывода, байтовых массивов и буферов друг в друга реализуется довольно просто. Методы для копирования потока ввода-вывода в байтовый массив или отправки содержимого байтового массива в поток ввода-вывода уже реализованы как часть .NET Framework. Класс `WindowsRuntimeBufferExtensions` предоставляет дополнительные возможности для преобразований буферов в байтовые массивы и наоборот. Он принадлежит пространству имен `System.Runtime.InteropServices.WindowsRuntime` и предоставляет собственный набор методов расширения,

в том числе AsBuffer (приводит экземпляр объекта типа Byte[] к типу IBuffer), AsStream (приводит экземпляр типа IBuffer к типу Stream) и ToArray (приводит экземпляр объекта типа IBuffer к типу Byte[]).

Сжатие данных

Хранение больших объемов данных может потребовать большого дискового пространства. Благодаря сжатию можно уменьшить общий объем хранимой информации, особым образом ее закодировав. Существуют два типа сжатия данных. *Сжатие без потерь* (lossless compression) обеспечивает сохранность исходных данных в неизменном виде. *Сжатие с потерями* (lossy compression) обеспечивает более высокие показатели производительности и сжатия, но некоторая часть исходной информации может быть утеряна. Этот вид сжатия часто используют при работе с изображениями, видеофайлами и звуковыми файлами, если абсолютно точное соответствие сжатой копии оригиналу не требуется.

WinRT предоставляет классы Compressor и Decompressor для работы со сжатыми данными. В проекте Compression есть работающий пример, реализующий сжатие и последующую декомпрессию потока данных. Проект содержит текстовый файл размером почти 100 Кбайт, загружает этот текст и выводит его в диалоговом окне, указывая размер в байтах. Затем можно щелкнуть на одной кнопке для сжатия текста, и на другой — для его декомпрессии.

Сжатие данных выполняется в несколько этапов. Сначала открывается локальный файл для записи сжатого текста. Текст можно закодировать различными способами, поэтому сначала используется класс Encoding для преобразования текста в байтовый массив формата UTF8:

```
var storage = await ApplicationData.Current.LocalFolder
  .CreateFileAsync("compressed.zip",
  CreationCollisionOption.ReplaceExisting);
var bytes = Encoding.UTF8.GetBytes(_text);
```

Ранее в этой главе вы узнали, как найти папку, которая содержит данные для текущих приложения и пользователя. Вы можете просмотреть содержимое этой папки в учебном приложении, чтобы увидеть сжатый файл, который появится там после щелчка на соответствующей кнопке. Файл хранится с расширением .zip, чтобы показать, что это сжатый файл, но на самом деле его формат отличается от формата ZIP, поэтому вы не сможете распаковать его в проводнике.

КОДИРОВКА

Тексты и отдельные символы при использовании различных региональных параметров и языков хранятся в виде последовательностей битов и байтов. То, как будут декодированы эти двоичные данные, зависит от кодировок. Одна из наиболее ранних схем кодировки называется ASCII и применяет для кодирования текстов 7-битные последовательности. Более современными схемами являются UTF-8 и UTF-16. В первой используются 8-битные последовательности, при необходимости для кодирования одного символа может быть задействовано до трех таких байтов; во второй — последовательности из 16-битных целых. .NET Framework предоставляет класс Encoding, который упрощает работу с данными различных форматов. Подробности об этом классе вы можете узнать на странице http://msdn.microsoft.com/ru-ru/library/86hf4sb8(v=vs.110).aspx.

Приведенный далее код открывает файл для записи, создает экземпляр класса `Compressor` и записывает байты. Затем осуществляется завершение операции сжатия и закрытие всех участвующих в ней потоков:

```
using (var stream = await storage.OpenStreamForWriteAsync())
{
  var compressor = new Compressor(stream.AsOutputStream());
  await compressor.WriteAsync(bytes.AsBuffer());
  await compressor.FinishAsync();
}
```

После завершения операции сжатия данные снова считываются с диска, чтобы вывести их объем. Вы обнаружите, что алгоритм сжатия, используемый по умолчанию, позволяет почти в два раза уменьшить объем файла. В операции декомпрессии используется класс `Decompressor`. Он выполняет обратные действия и помещает полученные данные в буфер (затем выполняется их сохранение на диске, и вы можете проверить результаты этой операции).

```
var decompressor = new Decompressor(stream.AsInputStream());
var bytes = new Byte[100000];
var buffer = bytes.AsBuffer();
var buf = await decompressor.ReadAsync(buffer, 999999,
  InputStreamOptions.None);
```

При создании классов для работы со сжатыми данными вы можете передать параметр, задающий алгоритм сжатия. В табл. 6.4 приведено описание этих алгоритмов.

Таблица 6.4. Алгоритмы сжатия данных

Члены перечисления CompressAlgorithm	Описание
InvalidAlgorithm	Представляет недоступный алгоритм сжатия. Используется при тестировании, чтобы инициировать исключение
NullAlgorithm	Представляет алгоритм, который передает данные в буфер без сжатия. Используется в основном для тестирования
Mszip	Представляет алгоритм MSZIP
Xpress	Представляет алгоритм XPRESS
XpressHuff	Представляет алгоритм XPRESS с кодированием по методу Хаффмана
Lzms	Представляет алгоритм LZMS

Благодаря среде выполнения Windows работа со сжатыми данными организована просто и понятно. Используйте сжатие данных в тех случаях, когда у вас имеются большие объемы данных, но вам приходится экономить необходимое приложению дисковое пространство. Учитывайте, что сжатие данных замедляет операции сохранения информации, поэтому не забудьте предварительно поэкспериментировать, чтобы найти алгоритм, обеспечивающий наилучшие показатели сжатия и производительности для данных того типа, сохранение которых вы выполняете. Помните и о том, что при декомпрессии данных вы должны указать тот же алгоритм, что и при сжатии.

Шифрование и цифровая подпись

Многие приложения хранят конфиденциальные данные, которые следует зашифровать, чтобы скрыть от чужих глаз. Это могут быть некие сведения о пользователе или собственные данные приложения. Некоторые данные может понадобиться подписать с помощью цифровой подписи. При этом генерируется специальный хэш-код данных, который представляет уникальную сигнатуру. Если исходные данные искажены или подделаны, сигнатура данных изменится. Вы можете сравнить полученную сигнатуру с исходной, чтобы определить, не изменены ли данные каким-либо образом.

Среда выполнения Windows с помощью класса `CryptographicEngine` поддерживает шифрование данных и создание цифровой подписи. Этот класс предоставляет средства для шифрования, дешифрования, создания циф-

ровых подписей и их проверки. В проекте EncryptionSigning есть несколько простых примеров выполнения этих операций. Основной программный код расположен в файле MainPage.xaml.cs.

Для выполнения шифрования и дешифрования нужен особый *ключ* (key). Рассматривайте этот ключ как пароль для шифрования и дешифрования. Существуют ключи двух видов. Проще всего так называемый *симметричный ключ* (symmetric key), который подразумевает использование одного и того же пароля, или «секретного» ключа, как для шифрования, так и для дешифрования информации.

Для того чтобы создать такой ключ, вы можете воспользоваться классом `SymmetricKeyAlgorithmProvider`. Класс инициализируют, вызывая метод `OpenAlgoritm` с именем выбранного для шифрования алгоритма. Для создания секретного ключа вызывают метод `CreateSymmetricKey`. Тот же самый ключ потребуется позже для дешифрования данных. Список доступных алгоритмов можно найти в MSDN-документации на странице http://msdn.microsoft.com/ru-ru/library/windows/apps/windows.security.cryptography.core.symmetrickeyalgorithmprovider.openalgorithm.aspx.

В нашем примере для шифрования и дешифрования данных применяется потоковый шифр **RC4**. Пользователю предлагается выбрать один из двух паролей, затем пароль повторяется 100 раз для заполнения буфера. Вы можете использовать в качестве ключа любой источник данных, который может быть преобразован в байтовый массив. В код включена вспомогательная утилита, предназначенная для преобразования строки в экземпляр `IBuffer`:

```
var buffer = CryptographicBuffer
  .ConvertStringToBinary(str.Trim(),
  BinaryStringEncoding.Utf8);
return buffer;
```

Класс `CryptographicBuffer` предоставляет набор вспомогательных методов для выполнения шифрования, дешифрования и создания цифровой подписи. Он поддерживает сравнение двух буферов, преобразование данных из строкового формата в формат двоичных массивов с использованием различных кодировок, декодирование и кодирование по алгоритму **Base64**, а также создание буферов, заполненных случайными данными. В данном примере он служит для кодирования строки, возвращаемой в буфер, в формат UTF8.

При наличии вспомогательного метода код, с помощью которого создается ключ, выглядит следующим образом:

```
var result = await GetPassword();
var provider = SymmetricKeyAlgorithmProvider.OpenAlgorithm("RC4");
var key = provider.CreateSymmetricKey(AsBuffer(result));
```

BASE64

Base64 — это схема кодирования, которая поддерживает преобразование двоичных данных в формат ASCII, что позволяет передавать двоичные данные в среде, поддерживающей лишь строковые данные и текст. Если вам нужно сохранить в формате JSON битовый массив, представляющий изображение, можете закодировать его по алгоритму Base64, а затем декодировать при десериализации JSON-строки. Размер закодированного текста всегда будет больше размера исходного изображения, так как при преобразовании в формат ASCII требуется, чтобы в байте, соответствующем символу, были использованы не все доступные биты. Обычно 3 байта (24 бита) исходных данных представляют в виде 4 байтов (32 бита) ASCII-кода. Поэтому Base64 иногда называют кодировкой 3 в 4.

Когда ключ создан, закодировать с его помощью исходный текст очень просто. Поскольку результат закодирован по алгоритму Base64, его можно вывести на экран в элементе `TextBlock` в правом столбце:

```
var encrypted = CryptographicEngine.Encrypt(key,
  AsBuffer(BigTextBox.Text), null);
_encrypted = encrypted.ToArray();
BigTextBlock.Text = CryptographicBuffer
  .EncodeToBase64String(encrypted);
```

После того как текст зашифрован, становится доступной кнопка для его дешифрования. Пользователю снова предлагается выбрать пароль, на этот раз — для дешифрования данных. Если пользователь выберет пароль, который отличается от такового для шифрования, дешифрование выполнить не удастся или его результат окажется нечитабельным. При дешифровании ключ получают точно так же, как и при шифровании, а затем просто вызывают метод `Decrypt` класса `CryptographicEngine`:

```
var decrypted = CryptographicEngine.Decrypt(key,
  _encrypted.AsBuffer(), null);
BigTextBox.Text = AsText(decrypted).Trim();
```

Помимо симметричного ключа шифровать данные можно с помощью *асимметричного ключа* (asymmetric key). Для создания такого ключа применяется класс `AsymmetricKeyAlgorithmProvider`. В этом случае для шифрования и дешифрования требуются два разных ключа: «открытый» и «секретный». Данные шифруются с помощью вашего конфиденциального секретного ключа, а для дешифрования вам нужно предоставить открытый ключ. Это позволяет третьей стороне успешно дешифровать данные, в то время как ваш секретный ключ не известен никому, кроме вас.

Узнать подробности об асимметричных ключах и найти примеры кода вы можете на странице http://msdn.microsoft.com/ru-ru/library/windows/apps/windows.security.cryptography.core.asymmetrickeyalgorithmprovider.openalgorithm.aspx.

Ключи, которые предназначены для создания цифровой подписи, можно создать с помощью класса `MacAlgorithmProvider`. Этот класс представляет код аутентификации сообщений (Message Authentication Code, MAC). Вы можете создать ключ, используя любой из популярных алгоритмов, в том числе MD5 (Message-Digest Algorithm), SHA (Secure Hash Algorithm) и CMAC (Cipher-based MAC). Данные ключи генерируются, в целом, так же, как и ключи шифрования. В нашем примере при создании ключа для цифровой подписи используется пароль, заданный по умолчанию:

```
var provider = MacAlgorithmProvider.OpenAlgorithm("HMAC_SHA256");
var key = provider.CreateKey(
  AsBuffer(MakeBigPassword(PASSWORD1)));
```

При выполнении цифровой подписи, так же, как и при шифровании, создается буфер. Разница заключается в том, что при шифровании данные из буфера можно использовать для дешифрования сообщения и восстановления его оригинала. В то же время восстановить сообщение из данных цифровой подписи нельзя. Цифровую подпись можно лишь передать вместе с сообщением в специальную функцию, которая позволит определить, не подделано ли сообщение.

Цифровая подпись создается вызовом метода `Sign` класса `CryptographicEngine`:

```
_signature = CryptographicEngine.Sign(key,
  AsBuffer(BigTextBox.Text)).ToArray();
```

Проверить цифровую подпись можно, вызвав функцию `VerifySignature`, которая вернет значение `true` (истина), если текст после создания цифровой подписи не менялся:

```
var result = CryptographicEngine.VerifySignature(key,
  AsBuffer(BigTextBox.Text),
  _signature.AsBuffer());
```

Для того чтобы увидеть, как это работает, запустите учебное приложение и щелкните на кнопке Sign (Подписать). После этого щелкните на кнопке Verify (Проверить), чтобы удостовериться в том, что текст не был изменен. Затем добавьте пробел или любой другой символ в текст в левой колонке и снова щелкните на кнопке Verify (Проверить). На этот раз вы увидите сообщение о том, что данный текст не соответствует оригиналу.

Windows 8 предоставляет разработчику набор мощных алгоритмов для шифрования, дешифрования и создания цифровой подписи. Эти задачи решаются достаточно просто с помощью классов `CryptographicEngine` и `CryptographicBuffer`, а также классов для работы с ключами. Используйте шифрования для защиты внутренних данных приложения и данных, передаваемых через Интернет. Применяйте цифровые подписи, проверяя, что данные, передаваемые по общедоступным сетям, не были подделаны при передаче.

Веб-службы

Веб-службы — это инструмент взаимодействия устройств через Интернет. Например, стандартный протокол для организации такого взаимодействия — простой протокол доступа к объектам (Simple Object Access Protocol, SOAP), разработанный в 1998 году. Если вы работали с SOAP, то знаете, что он не так уж и прост, поэтому сейчас быстро набирает популярность протокол передачи репрезентативных состояний (Representational State Transfer, REST).

Веб-службы играют важную роль во взаимодействии приложений. Многие системы уровня предприятия предоставляют веб-сервисы для клиентских приложений, таких как приложения для Windows 8. Одно из преимуществ протокола SOAP заключается в том, что он предоставляет механизм для получения сведений о структуре API веб-сервиса с помощью языка описания веб-сервисов (Web Services Description Language, WSDL). Ознакомиться со спецификацией WSDL вы можете на странице http://www.w3.org/TR/wsdl.

Откройте проект WeatherService, чтобы взглянуть на пример веб-служб. В примере используется бесплатная веб-служба, предоставляющая информацию о погоде. Для подключения к службе достаточно щелкнуть правой кнопкой мыши на узле References в обозревателе решений (Solution Explorer), выбрать в появившемся меню команду Add Service Reference (Добавить ссылку на службу) и ввести URL-адрес службы. Результат показан на рис. 6.4.

После добавления службы автоматически создается клиентский модуль доступа к ней. Модуль доступа берет на себя решение всех задач, необходимых для выполнения запросов к API службы и для передачи в приложение данных, полученных от службы. Все это представлено в виде асинхронных реализаций интерфейсов сервера. В нашем примере пользователю предлагается ввести почтовый индекс (zip-код). Когда пользователь щелкает на кнопке, проводится проверка введенного индекса и, если он введен без ошибок, осуществляется его передача веб-службе. Приведенные далее

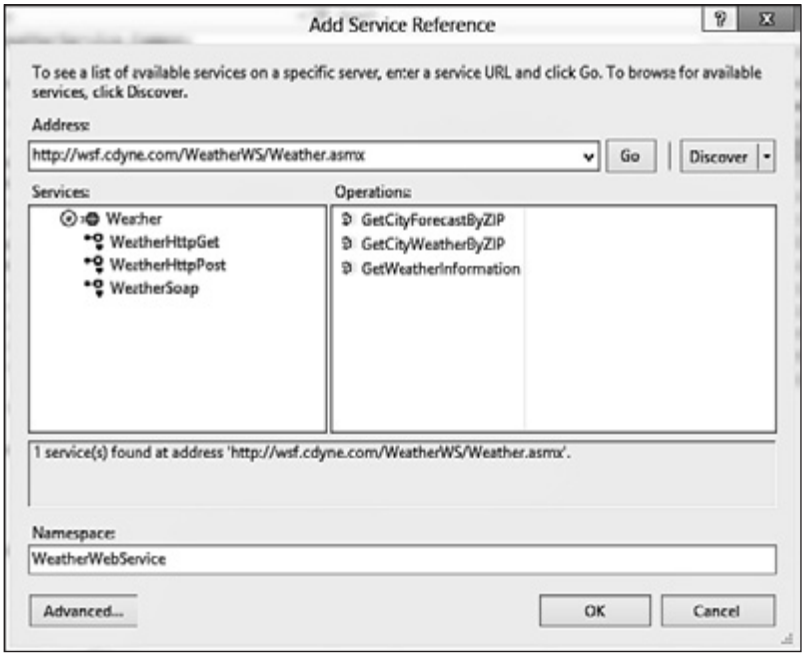


Рис. 6.4. Добавление в проект ссылки на веб-службу на основе SOAP

строки кода — это все, что нужно для создания модуля доступа для подключения к веб-службе и вызова ее с указанием почтового индекса (обратите внимание на то, что в соответствии с соглашением об именовании в конце имени метода указывается ключевое слово `Async`):

```
var client = new WeatherWebService.WeatherSoapClient();
var result = await client.GetCityForecastByZIPAsync(
  zip.ToString());
```

Если данный вызов завершится неудачно, появится диалоговое окно, сообщающее об этом. В противном случае полученные результаты будут с помощью механизма привязки данных показаны в сетке, на основе которой построен макет страницы приложения. Вот как выполняется привязка полученного от веб-службы результата к сетке:

```
ResultsGrid.DataContext = result;
```

А вот как выглядит XAML-разметка, предназначенная для вывода данных о городе и штате:

```
<StackPanel Orientation="Horizontal">
  <TextBlock Text="{Binding City}"/>
```

```
  <TextBlock Text=","/>
  <TextBlock Text="{Binding State}"/>
</StackPanel>
```

В листинге 6.3 показана вся XAML-разметка для отдельной позиции прогноза погоды. Поле «description» (описание) в данной разметке представлено так неслучайно — именно так его предоставлял веб-сервис в момент создания данного примера.

Листинг 6.3. Привязка данных результатов, полученных от погодной веб-службы

```
<ListView Grid.Row="1" ItemsSource="{Binding ForecastResult}">
  <ListView.ItemTemplate>
    <DataTemplate>
      <StackPanel Orientation="Horizontal">
        <TextBlock Text="{Binding Date, Converter={StaticResource
             ↳ ConvertDate}}"
          Width="200"/>
        <TextBlock Text="{Binding Desciption}" Width="150"
             ↳ Margin="5 0 0 0"/>
        <Image Source="{Binding Desciption,
             ↳ Converter={StaticResource ConvertImage}}"/>
        <TextBlock Text="{Binding Temperatures.MorningLow}"
             ↳ Margin="5 0 0 0" Width="50"/>
        <TextBlock Text="{Binding Temperatures.DaytimeHigh}"
             ↳ Margin="5 0 0 0" Width="50"/>
      </StackPanel>
    </DataTemplate>
  </ListView.ItemTemplate>
</ListView>
```

В документации к рассматриваемой погодной веб-службе есть описание значков, которые соответствуют описаниям. Класс `ImageConverter` преобразует описания погодных условий в пути к графическим файлам, которые он может возвратить для их вывода в интерфейсе приложения:

```
var filename =
  string.Format("ms-appx:///Assets/{0}.gif",
    ((string) value).Replace(" ", string.Empty).ToLower());
return new BitmapImage(new Uri(filename, UriKind.Absolute));
```

На рис. 6.5 показан результат запроса прогноза погоды по почтовому индексу для моего города (Вудсток, штат Джорджия):

Рис. 6.5. Прогноз погоды для города Вудсток, штат Джорджия

Поддержка протокола OData

Open Data Procotol (OData) — это веб-протокол, который используется для запроса и обновления данных. Это прикладной программный интерфейс на основе технологии REST, построенный поверх протокола Atom и использующий для передачи данных формат JSON или XML. Подробности об OData можно узнать на странице http://www.odata.org/.

Приложения для Windows 8 имеют естественную поддержку OData-клиентов, достаточно лишь загрузить и установить клиент со страницы http://go.microsoft.com/fwlink/?LinkId=253653.

Для доступа к OData-службам достаточно добавить ссылку на службу — так же, как мы делали ранее для веб-служб на основе SOAP. Популярная OData-служба, которой часто пользуются для демонстрации возможностей данного протокола, — каталог видеоматериалов Netflix. Вы можете самостоятельно взглянуть на этот сервис, перейдя по ссылке http://odata.netflix.com/catalog/ в браузере.

В большинстве браузеров вы увидите XML-документ с различными тегами, представляющими коллекции, которые вы можете просматривать. Например, наличие коллекции Titles говорит нам о том, что вы можете просматривать канал, содержащий список заголовков и кратких описаний материалов сервиса, доступный на странице http://odata.netflix.com/catalog/Titles.

Проект Netflix предлагает простую демонстрацию использования OData-канала. Основной URL-адрес нужно добавить в проект в виде ссылки на службу, так же, как мы делали это в предыдущем примере. Первый шаг в применении службы заключается в создании модуля доступа к ней. Для этого служит класс, который генерируется автоматически при добавлении службы и передачи ей URL-адреса:

```
var netflix =
  new NetflixCatalog(
    new Uri(
      "http://odata.netflix.com/Catalog/",
      UriKind.Absolute));
```

Далее осуществляется настройка коллекции для хранения результатов, которые будут получены с помощью OData-запроса. Для этого предназначается специальный класс `DataServiceCollection`:

```
private DataServiceCollection<Title> _collection;
...
_collection = new DataServiceCollection<Title>(netflix);
TitleGrid.ItemsSource = _collection;
```

И наконец, создается запрос для фильтрации данных. Этот запрос передается модулю доступа и загружает результаты в коллекцию. В данном примере запрос выбирает первые 100 заголовков, которые начинаются с буквы «Y», и сортирет их в порядке убывания рейтинга:

```
var query = (from t in netflix.Titles
             where t.Name.StartsWith("Y")
             orderby t.Rating descending
             select t).Take(100);
_collection.LoadAsync(query);
```

В итоге, когда поступят данные, у вас есть возможность запросить дополнительные наборы данных. Для этого коллекция проверяет на наличие *продолжения* (continuation). Если продолжение существует, вы можете запросить у сервиса следующую порцию данных. В результате данные можно получать сравнительно небольшими порциями, а не загружать единовременно огромный объем информации:

```
if (_collection.Continuation != null)
{
  _collection.LoadNextPartialSetAsync();
}
```

Запустите учебное приложение. Вы увидите, как заголовки и изображения начинают асинхронно выводиться в сетке, которую можно прокручивать. Как и в предыдущем примере, данные, возвращенные веб-службой, напрямую привязаны к сетке:

```
<Image Stretch="Uniform" Width="150" Height="150">
  <Image.Source>
    <BitmapImage UriSource="{Binding BoxArt.LargeUrl}"/>
  </Image.Source>
</Image>
<TextBlock Text="{Binding Name}" Grid.Row="1"/>
```

Среда разработки приложений для Windows 8 упрощает работу с веб-службами и получение данных из внешних источников. Многие существующие приложения предоставляют доступ к веб-службам в виде SOAP-, REST- или OData-каналов. Благодаря встроенной поддержке доступа к этим каналам и работы с ними можно создавать такие приложения для Windows 8, которые реализуют существующую функциональность посредством веб-служб.

Выводы

В этой главе рассмотрены различные подходы к работе с данными, которые доступны при разработке приложений для Windows 8. Вы узнали, как сохранять и получать данные из файловых хранилищ, работать с данными через Интернет, транслировать их с помощью RSS- и Atom-каналов. Вы познакомились со встроенными инструментами, которые упрощают операции шифрования данных и создание цифровой подписи. Наконец, вы увидели, как просто подключаться к существующим веб-службам, поддерживающим протоколы SOAP и OData, создавая модули доступа и асинхронно получая данные из внешних прикладных программных интерфейсов.

В следующей главе вы узнаете, как, применяя плитки и уведомления, сохранять приложение активным даже тогда, когда оно не запущено. Плитки на начальном экране предоставляют пользователю краткую информацию, которая постоянно обновляется, хотя соответствующие приложения не запущены. Уведомления могут быть инициированы внутренними механизмами приложения либо исходить из внешних источников. Их назначение — оповещать пользователя о важных событиях и предоставлять ссылки для доступа к определенным страницам приложения.

Плитки и уведомления

7

Приложения для Windows 8 могут быть активными и оставаться на связи даже тогда, когда они не выполняются в качестве приложений переднего плана. Это становится очевидным, если посмотреть на начальный экран Windows 8. В отличие от скучных рядов статичных значков, с которыми вы, вероятно, сталкивались ранее, начальный экран Windows 8 активный и динамичный. Плитка погодного приложения выводит сведения о текущей температуре и изображение, по которому можно с одного взгляда оценить погодные условия. Плитки приложений для социальных сетей воспроизводят последние твиты, которые могут быть вам интересны. Плитка приложения электронной почты показывает количество непрочитанных сообщений, на ней же можно увидеть их темы. Если у вас есть мобильный телефон с ОС Windows Phone, вы наверняка уже знакомы с концепцией активных плиток и каждый день ими пользуетесь.

Всю эту функциональность обеспечивают плитки. Плитки встроены в среду выполнения Windows. Существуют прикладные программные интерфейсы, призванные упростить передачу плиткам информации в различных форматах и шаблонах. Очень важно и то, что вы можете обновлять плитки и отправлять уведомления даже тогда, когда ваше приложение не является приложением переднего плана или вообще не запущено. Это позволяет постоянно сохранять приложение активным и адекватным.

Обычные плитки

С обычными плитками вы уже знакомы. В редакторе манифеста приложения, на вкладке Application UI (Интерфейс приложения) можно задать квадратный значок размером 150×150 пикселов и широкий значок размером 310×150 пикселов. Эти значки используются в качестве статичных плиток, назначаемых приложению по умолчанию.

Широкий значок задействовать не обязательно, но когда он задан, пользователь может выбирать между прямоугольной и квадратной плиткой приложения. Широкий значок удобнее, так как он предоставляет больше места для размещения информации при обновлении плитки. Пример настройки плиток в редакторе манифеста вы можете видеть на рис. 7.1.

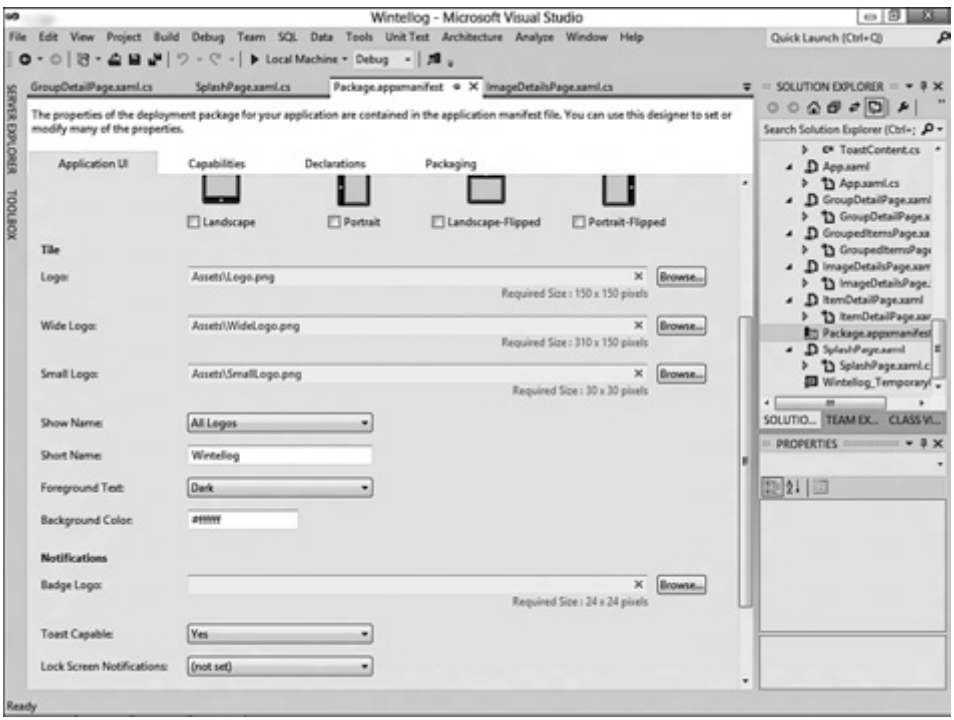


Рис. 7.1. Настройка значков для плиток приложения

Оба вида плиток поддерживают обновление. Активные плитки можно обновлять либо из самого приложения, либо из облачных сервисов, перезаписывая контент базового шаблона. Плитки могут перекрывать базовый шаблон источниками простой информации, такими как индикаторы и индикаторные счетчики, либо полностью заменять базовый шаблон. Для выбора доступно множество шаблонов, включая графические, текстовые и комбинированные.

Активные плитки

Активные плитки позволяют сформировать у пользователя ощущение, что приложение выполняет некоторые действия даже тогда, когда оно не запущено. Среда выполнения Windows упрощает настройку формата плиток и вывода на них какой-либо информации. Сделать это не сложнее, чем подготовить необходимый XML-код и вызвать метод вспомогательного класса. Пожалуй, самое сложное здесь — выбрать, какой именно формат плитки использовать.

Со страницы http://code.msdn.microsoft.com/windowsapps/App-tiles-and-badges-sample-5fc49148 вы можете загрузить приложение, выводящее на экран список доступных форматов плиток.

Загрузите исходный код, скомпилируйте приложение и запустите его. Вам будет представлено несколько различных сценариев. Выбрав пятый сценарий, вы сможете просматривать доступные шаблоны, задавать изображения и текст для обновления плитки. На рис. 7.2 показана копия экрана нашего учебного приложения, демонстрируя разновидности доступных форматов плиток. Приложение также содержит набор расширений плиток, вы можете воспользоваться ими в собственных приложениях, чтобы упростить настройку плиток и индикаторов.

Предопределенные шаблоны содержат изображения, текст и их комбинации. «Обзорные» шаблоны поддерживают анимацию и переключение с одного формата на другой. Для использования каждого шаблона нужно задать текст и путь к изображению (если шаблон предоставляет изображения) в XML-документе и передать этот документ среде выполнения Windows.

Подробный список шаблонов плиток, примеры изображений и соответствующего им XML-кода можно найти на странице http://msdn.microsoft.com/ru-ru/library/windows/apps/hh761491.aspx.

Загрузите обновленное приложение Wintellog2 с веб-сайта http://windows8-applications.codeplex.com/.

Для данной главы пример был обновлен по следующим позициям:

- Введен класс `ImageUriManager`, позволяющий добиться лучшего разрешения изображений для каждого поста блога.
- Добавлена панель приложения для просмотра изображений, перехода к домашней странице и открытия постов в браузере.
- Добавлен на главную страницу инструмент семантического масштабирования для быстрого доступа к различным блогам.

Рис. 7.2. Различные шаблоны плиток

В приложении все еще не реализованы некоторые важные механизмы, такие как интеграция с чудо-кнопками Search (Поиск) и Share (Общий доступ). Эти возможности будут реализованы в главе 8. Однако версия приложения для данной главы получила плитки и индикаторы. Кроме того, в нем реализован новый механизм, позволяющий закрепить в стартовом меню ссылку на какой-либо блог или на отдельный пост блога. Обновление плиток производится с помощью класса `SplashPage`.

После загрузки списка постов из кэша и из Интернета приложение запрашивает пять последних постов и сортирует их в восходящем порядке по дате публикации:

```
var query = from i in
            (from g in list
             from i in g.Items
             orderby i.PostDate descending
             select i).Take(5)
    orderby i.PostDate
    select i;
```

Обновлять плитки можно двумя способами. В режиме, применяемом по умолчанию, производится замена текущей плитки на ту, которую вы указали последней. Другой подход предусматривает постановку плиток в очередь уведомлений. Когда обновления производятся в этом режиме, на плитке циклически воспроизводятся пять последних обновлений. Это позволяет вам размещать на плитках несколько фрагментов данных, таких как заголовки непрочитанных сообщений, или, в случае с нашим примером, заголовки пяти последних постов.

WinRT предоставляет класс `TileUpdaterManager`, который используется для настройки плиток. Следующий код подключает очередь уведомлений:

```
TileUpdateManager.CreateTileUpdaterForApplication()
  .EnableNotificationQueue(true);
```

Очередь продвигается для каждого поста, кроме того, с использованием примеров кода для Windows 8 созданы две плитки. Квадратная плитка обеспечивает быстрый просмотр изображения и содержит текст заголовка:

```
var squareTile = new TileSquarePeekImageAndText04();
squareTile.TextBodyWrap.Text = item.Title;
squareTile.Image.Alt = item.Title;
squareTile.Image.Src = item.DefaultImageUri.ToString();
```

При создании прямоугольных плиток всегда следует создавать и квадратные — на тот случай, если пользователь решит, что на начальном экране должна располагаться квадратная плитка. Для того чтобы изменить размер плитки, достаточно выделить ее на начальном экране, после чего появится возможность сделать плитку меньше или больше в зависимости от текущих размеров плитки и поддержки плиток различного размера приложением. Панель приложения, которая позволяет выполнить такую настройку, показана на рис. 7.3.

Вот код для широкой плитки, который ссылается на квадратную плитку в файле SplashScreen.xaml.cs:

```
var wideTile = new TileWideSmallImageAndText03();
wideTile.SquareContent = squareTile;
wideTile.Image.Alt = item.Title;
wideTile.Image.Src = item.DefaultImageUri.ToString();
wideTile.TextBodyWrap.Text = item.Title;
```

Для того чтобы увидеть, как классы расширений помогают создавать XML-код для плиток, установите точку останова после команды создания широкой плитки на строке, которая генерирует уведомление:

```
var notification = wideTile.CreateNotification();
```

Рис. 7.3. Настройка плиток на начальном экране

Запустите приложение и позвольте сработать точке останова. Откройте окно интерпретации (Immediate Window), либо выбрав в меню команду Debug ▸ Window ▸ Immediate Window (Отладка ▸ Окна ▸ Интерпретация), либо нажав комбинацию клавиш Ctrl+Alt+I. Это окно позволяет вводить инструкции и переменные для вычисления значений во время выполнения приложения. В окне интерпретации введите слово `wideTile` (регистр символов имеет значение) и нажмите клавишу Enter. Вы увидите дамп класса, как показано на рис. 7.4. Можете использовать команду `wideTile.GetContent()` для вывода только XML-кода.

Как видите, используемый XML-код достаточно прост. Он содержит привязки, относящиеся к различным размерам плиток. Привязка задает шаблон и затем контент для шаблона. Теги изображений и текста имеют идентификаторы для поддержки плиток, которые позволяют выводить

несколько записей. Для того чтобы показать изображение на плитке, можно воспользоваться одним из трех способов.

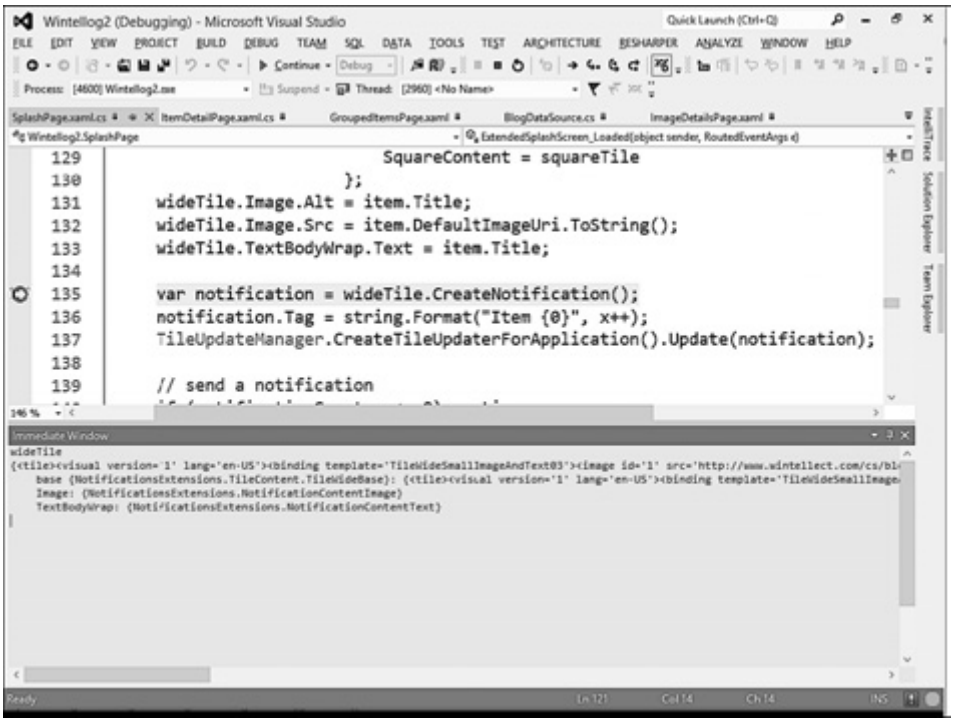


Рис. 7.4. Окно интерпретации в режиме отладки

Листинг 7.1. XML-код для широкой плитки с квадратной плиткой

```
<tile>
  <visual version='1' lang='en-US'>
    <binding template='TileWideSmallImageAndText03'>
      <image id='1' src='http://lh6.ggpht.com/-V8-lch3ip3U/
          ↳ T33GescOOHI/AAAAAAAAAps/vuQXH0JYqz0/
           slbookcover%25255B3%25255D.png?imgmax=800'
  alt='Designing Silverlight Business Applications
          ↳ Officially Released'/>
      <text id='1'>Designing Silverlight Business Applications
          ↳ Officially Released</text>
    </binding>
    <binding template='TileSquarePeekImageAndText04'>
      <image id='1' src='http://lh6.ggpht.com/-V8-lch3ip3U/
          ↳   T33GescOOHI/AAAAAAAAAps/vuQXH0JYqz0/
           slbookcover%25255B3%25255D.png?imgmax=800'
```

продолжение ⇩

Листинг 7.1 *(продолжение)*

```
  alt='Designing Silverlight Business Applications Officially
        ↳ Released'/>
      <text id='1'>Designing Silverlight Business Applications
        ↳ Officially Released</text>
    </binding>
  </visual>
</tile>
```

Для того чтобы сослаться на изображения, хранящиеся в Интернете, используют префикс `http://`. Кроме того, можно задействовать изображения, внедренные в пакет вашего приложения для Windows 8, указав префикс `ms-appx:///`, например значок для приложения Wintellog доступен по адресу `ms-appx:///Assets/Logo.png`. И, наконец, к изображениям, которые хранятся в локальном хранилище данных приложения, можно обратиться с помощью префикса `ms-appdata:///local/`. Вы можете загрузить изображения из Интернета и обращаться к ним подобным образом при отсутствии доступа в Интернет. Максимальный размер изображений для плиток не может превышать 150 Кбайт.

Выбор верного размера плитки для приложения зависит от нескольких вещей. Квадратная плитка предназначена для вывода кратких сообщений и уведомлений, которые обновляются не слишком часто. Использовать прямоугольные плитки рекомендуется тогда, когда обновлять их контент планируется достаточно часто. Этот размер соответствует принятому в Windows 8 лозунгу «всегда на связи, всегда активен».

Вот еще некоторые рекомендации компании Microsoft по применению плиток:

- Не используйте несколько плиток, чтобы обеспечить доступ к отдельным частям вашего приложения. Для этой цели лучше подходят вспомогательные плитки (о них мы поговорим позже).
- Не используйте плитки, чтобы предлагать пользователю установить дополнения к вашему приложению.
- Не создавайте плитки, предназначенные для решения проблем с приложением или для доступа к его параметрам. Для этих целей используйте чудо-кнопку Settings (Параметры), о которой подробно рассказывается в главе 8).
- Не помещайте на плитке призывов к запуску приложения. Это действие подразумевается наличием плитки на начальном экране.
- Не используйте плитки для вывода рекламы.

- Не выводите на плитке при ее обновлении относительные сведения о времени некоего события, например «10 минут назад», так как подобные сообщения быстро теряют актуальность. Вместо этого указывайте абсолютные значения даты и времени.

Полное руководство Microsoft по использованию плиток вы можете найти на странице http://msdn.microsoft.com/ru-ru/library/windows/apps/hh465403.aspx.

Когда вы активизируете очередь уведомлений, нужно учитывать наличие дополнительного свойства плитки, которое требуется задать. Это свойство для тега. Теги служат для идентификации отдельных позиций в очереди уведомлений. С помощью тегов вы можете обновить плитку в конкретной позиции, отправив новую плитку с тем же тегом. В учебном приложении определено пять позиций, в которые осуществляется запись пяти последних постов.

Индикаторы

В дополнение к концепции активных плиток, WinRT поддерживает идею уведомления пользователей посредством индикаторов. Индикаторы (badges) — это маленькие значки, которые выводятся на поверхности плитки. Обычно они служат для передачи некой обобщающей или статусной информации. Есть два типа индикаторов: числовые и графические. Числовые индикаторы показывают, например, количество новых записей. Графические индикаторы, или глифы, иллюстрируют состояние некоторого объекта (например, указывают на активность объекта, на то, является ли он новым) или о чем-то предупреждают. Индикаторы могут присутствовать как на квадратных, так и на прямоугольных плитках, располагаясь в их правом нижнем углу (исключая случаи, когда в системе используется язык с направлением письма справа налево).

СОВЕТ

В учебном приложении индикаторы призваны информировать о новых постах. Когда вы запускаете приложение впервые, вы видите цифровой индикатор количества загруженных постов. После этого вам может понадобиться протестировать функциональность индикатора даже при отсутствии новых постов. Для этого найдите папку локальных данных приложения (о том, как это сделать, мы говорили в предыдущей главе) и откройте папку любой из групп. Удалите несколько позиций из этой папки, чтобы заставить приложение заново загрузить посты. Когда вы запустите приложение в следующий раз, оно загрузит эти посты и обновит индикатор, чтобы сообщить о наличии новых позиций.

Цифровой индикатор может отображать любые числа от 1 до 99. В нашем приложении этот индикатор служит для вывода количества новых постов, загруженных с момента предыдущего запуска приложения.

Сначала вычисляется количество новых позиций. Их максимальное значение составляет 99:

```
var totalNew = list.Sum(g => g.NewItemCount);
if (totalNew > 99)
{
  totalNew = 99;
}
```

Затем производится обновление индикатора с использованием методов расширения из набора примеров кода для Windows 8, которые находятся в папке NotificationExtensions. Если имеются новые позиции, инициируется обновление числового индикатора:

```
if (totalNew > 0)
{
  var badgeContent = new BadgeNumericNotificationContent(
    (uint)totalNew);
  BadgeUpdateManager.CreateBadgeUpdaterForApplication()
    .Update(badgeContent.CreateNotification());
}
```

XML-код для этого весьма прост и понятен. Вот код индикатора, показывающего число 3:

```
<badge version="1" value="3"/>
```

Если новых позиций нет, индикатор просто очищается:

```
BadgeUpdateManager.CreateBadgeUpdaterForApplication().Clear();
```

Индикаторы могут быть также графическими. Их называют глифами (glyphs). Для работы с такими индикаторами имеется класс `BadgeGlyphNotificationContent`. Среда выполнения Windows предоставляет заранее заданный набор глифов. Вот их список:

- None (индикатор не отображается);
- Activity (выполнение задачи);
- Alert (оповещение);

- Available (доступен);
- Away (нет на месте);
- Busy (занят);
- New Message (новое сообщение);
- Paused (воспроизведение приостановлено);
- Playing (воспроизведение);
- Unavailable (недоступен);
- Error (ошибка);
- Attention (внимание).

Примеры изображений для каждого из глифов вы можете найти на странице http://msdn.microsoft.com/ru-ru/library/windows/apps/hh761458.aspx.

Кроме того, в Microsoft предлагается несколько рекомендаций, касающихся индикаторов:

- Числовые индикаторы следует использовать лишь тогда, когда сведения о количестве элементов имеют смысл. Если есть вероятность того, что придется выводить очень большие числовые значения, следует рассмотреть возможность использования графического индикатора.
- Числа на индикаторах должны соответствовать состоянию приложения на момент его последнего запуска, а не состоянию на момент его установки.
- Если ваше приложение поддерживает различные типы уведомлений, лучше использовать, например, для оповещения о новых сообщениях, графический, а не числовой индикатор, поскольку последний допускает неоднозначное толкование показателей.
- Используйте графические индикаторы только тогда, когда может меняться состояние, обозначаемое графическим индикатором.

Индикаторы работают в связке с механизмом обновления активных плиток. Учебное приложение наряду с демонстрацией на плитке сведений о самых последних постах выводит индикатор, показывающий количество новых постов. Подобная комбинация нового контента и информирующего об этом контенте индикатора — обычная практика. Вы можете внести в программу изменение, чтобы показывать не пять последних постов, а только новые посты. Тогда контент плитки будет соответствовать показаниям индикатора.

Вспомогательные плитки

Запустите наше учебное приложение и перейдите на страницу просмотра группы или отдельной позиции. Откройте панель приложения, и вы увидите кнопку Pin to Start (Закрепить на начальном экране), как показано на рис. 7.5.

Рис. 7.5. Кнопка для закрепления на начальном экране дополнительной плитки приложения, соответствующей конкретному блогу

При активизации данной кнопки на начальном экране выводится новая плитка с логотипом приложения и заголовком блога. Когда вы щелкаете на этой плитке, приложение запускается и автоматически переходит к странице данного блога. То же самое происходит при закреплении на начальном экране вспомогательной плитки для отдельной позиции блога: запуск приложения с помощью этой плитки приводит к открытию страницы для просмотра данной позиции.

Вспомогательные плитки (secondary tiles) называются так потому, что они предлагают дополнительные способы запуска приложения в обход основной плитки. Они призваны упростить пользователю доступ к интересующему его контенту или помочь ему глубже погрузиться в определенные

места приложения. Вспомогательные плитки нельзя создать программно. Их может создавать лишь пользователь, то есть команда закрепления плитки приводит к вызову диалогового окна, которое и дает пользователю возможность создать плитку. Пользователь может также удалить плитку с начального экрана, выделив ее и выбрав в панели приложения команду Unpin from Start (Открепить от начального экрана).

Вспомогательные плитки должны быть квадратными, но могут иметь и прямоугольную форму. Они способны отображать уведомления и индикаторы (они должны соответствовать контенту плитки, например показывать только новые посты заданного блога). Вспомогательные плитки автоматически удаляются с начального экрана при удалении приложения.

Когда вы выбираете команду закрепления на начальном экране группы, выполняется следующий код:

```
var group = DefaultViewModel["Group"] as BlogGroup;
var title = string.Format("Blog: {0}", group.Title);
App.Instance.PinToStart(sender,
  string.Format("Wintellog.{0}", group.Id.GetHashCode()),
  title,
  title,
  string.Format("Group={0}", group.Id));
```

В первой строке мы получаем текущую группу, во второй — создаем дополнительную плитку. Метод `PinToStart` — это вспомогательный метод, который упрощает вывод пользователю запроса на создание вспомогательной плитки. Он принимает UI-элемент (соответствующую кнопку на панели приложения), который инициировал создание новой плитки, уникальный идентификатор плитки (в данном случае мы используем хэш-код идентификатора группы), краткое имя, выводимое имя и набор аргументов. Аргументы хранятся вместе с плиткой и передаются приложению при его запуске посредством плитки. Это позволяет приложению определить, какую именно страницу следует отобразить при запуске.

Вспомогательный метод создает несколько ссылок на внедренные ресурсы, использующиеся для плиток разных размеров:

```
var logo = new Uri("ms-appx:///Assets/Logo.png");
var smallLogo = new Uri("ms-appx:///Assets/SmallLogo.png");
var wideLogo = new Uri("ms-appx:///Assets/WideLogo.png");
```

Далее создается объект вспомогательной плитки и устанавливаются некоторые его свойства. Среда выполнения Windows предоставляет для этих целей класс, расположенный в пространстве имен `Windows.UI.StartScreen`:

```
var tile = new SecondaryTile(id, shortName, displayName, args,
  TileOptions.ShowNameOnLogo | TileOptions.ShowNameOnWideLogo,
  logo);
tile.ForegroundText = ForegroundText.Dark;
tile.SmallLogo = smallLogo;
tile.WideLogo = wideLogo;
```

Последний фрагмент кода принимает родительский элемент, запросивший создание плитки, и вычисляет позицию на экране. Эти данные передаются в особое диалоговое окно, называемое *всплывающим* (flyout). Оно помогает понять, где расположится новая плитка. Всплывающее окно — это простое диалоговое окно, появляющееся в некотором контексте и способное автоматически закрываться наподобие панели чудо-кнопок. Далее вызывается метод плитки, который запрашивает у пользователя подтверждение для размещения ее на начальном экране.

```
var element = sender as FrameworkElement;
var buttonTransform = element.TransformToVisual(null);
var point = buttonTransform.TransformPoint(new Point());
var rect = new Rect(point,
  new Size(element.ActualWidth, element.ActualHeight));
await tile.RequestCreateForSelectionAsync(rect,
  Windows.UI.Popups.Placement.Left);
```

На рис. 7.6 вы можете видеть результат прикосновения к кнопке Pin to Start (Закрепить на начальном экране). Обратите внимание, что всплывающее окно появилось левее и выше кнопки, которой коснулся пользователь. До закрепления плитки пользователь может отредактировать текст, который должен выводиться на плитке. Всплывающие окна — это важный атрибут пользовательского интерфейса Windows 8. Хотя для работы с ними не предусмотрено встроенных элементов управления, Тим Хейер (Tim Heuer) предложил набор инструментов, который называется Callisto (https://github.com/timheuer/callisto). В него входит и реализация данного элемента управления, которую вы можете использовать в своих проектах.

В данном случае аргументы, передаваемые плитке, включают в себя группу и ее идентификатор:

```
Group=http://feeds2.feedburner.com/CSharperImage
```

Когда расширенная заставка завершает загрузку всего необходимого контента, проверяются аргументы, переданные приложению. Если аргументы начинаются со слова `Group`, программа обрабатывает идентификатор группы и вызывает страницу просмотра группы с этим идентификатором:

```
if (_activationArgs.Arguments.StartsWith("Group"))
{
  var group = _activationArgs.Arguments.Split('=');
  rootFrame.Navigate(typeof (GroupDetailPage), group[1]);
}
```

Рис. 7.6. Всплывающее окно для закрепления дополнительной плитки на начальном экране

Приложение отзывается на запуск через вспомогательные плитки, обрабатывая аргументы, переданные ему при запуске. Если вы попытаетесь установить точку останова и изменить эти аргументы, то обнаружите, что они предназначены только для чтения. Как же отлаживать вспомогательные плитки? Это можно делать независимо от того, как было запущено приложение, посредством основной плитки или вспомогательной. Для начала щелкните правой кнопкой мыши на проекте приложения в обозревателе решений и в появившемся меню выберите пункт Properties (Свойства) или нажмите клавиши Alt+Enter. Будет открыто диалоговое окно свойств проекта. Перейдите на вкладку Debug (Отладка) и установите флажок Do not launch, but debug my code when it starts (Не запускать, а отлаживать мой код при открытии), как показано на рис. 7.7.

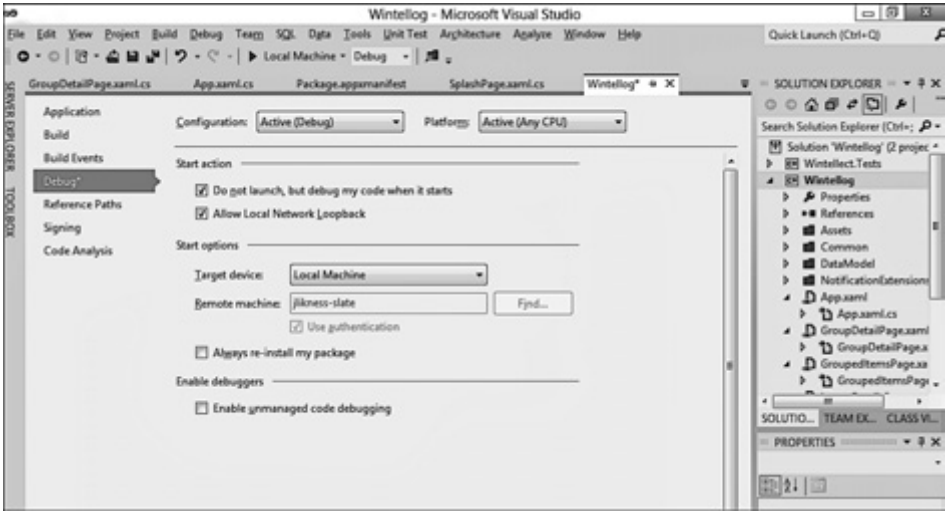


Рис. 7.7. Отладка приложения без его автоматического запуска

Теперь вы можете нажать клавишу F5, начав отладку. Запустится отладчик, но приложение не запустится. Нажмите клавишу Windows, чтобы открыть начальный экран, и найдите вспомогательную плитку приложения. Коснитесь ее, чтобы запустить приложение. Отладчик автоматически подключится к процессу приложения и остановит его в тех местах, где вы установили точки останова. Далее вы можете просматривать аргументы, переданные приложению вспомогательной плиткой, и отлаживать приложение, запущенное с ее помощью.

Всплывающие уведомления

Всплывающие уведомления позволяют отправлять пользователю короткие сообщения за пределы приложения. Хотя их отправка может быть инициирована приложением, обычно это делают внешние источники. В следующем разделе вы узнаете также о push-уведомлениях и о том, как обновлять данные приложения даже тогда, когда оно не запущено.

Пользователь полностью контролирует всплывающие уведомления. Когда пользователь открывает окно после щелчка на чудо-кнопке Settings (Параметры), там всегда присутствует команда Permissions (Разрешения). После активизации этой команды становится доступным список параметров, похожий на тот, который показан на рис. 7.8.

Уведомления можно как отправлять немедленно, так и планировать это в будущем. Например, приложение для работы с календарем могло бы

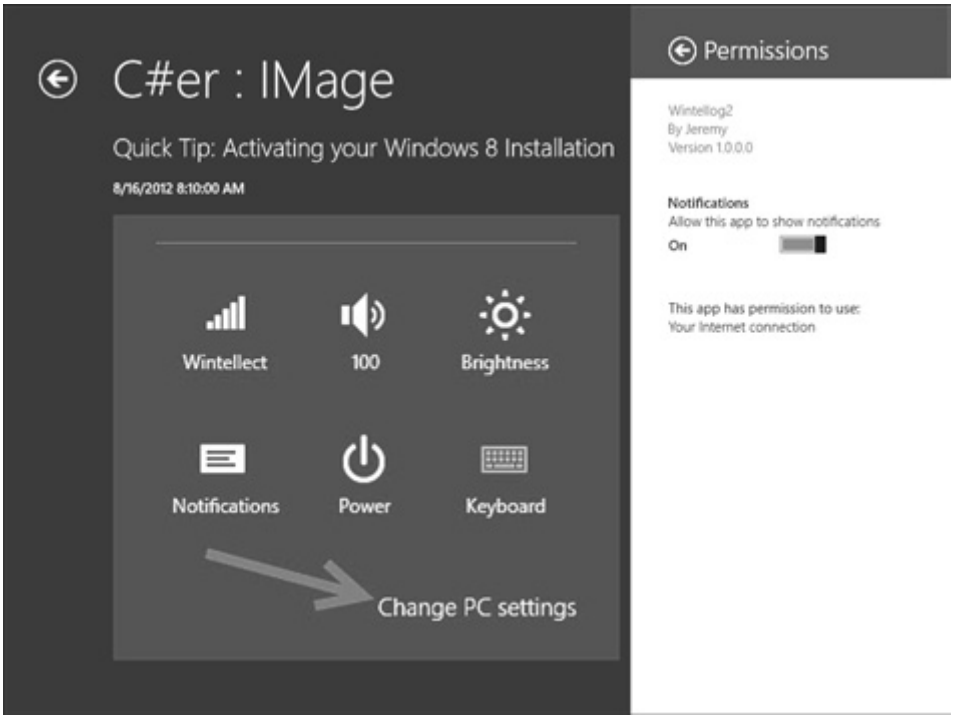


Рис. 7.8. Настройка уведомлений пользователем

напомнить о будущем событии. Хотя уведомление может планироваться для отправки из приложения, отложенный показ уведомлений можно запланировать и в ходе обычной работы приложения, они будут появляться даже в том случае, если приложение больше не работает. Неоднократно упоминаемое нами приложение использует эту возможность для планирования всплывающих уведомлений о новых позициях. Однако это вовсе не пример того, каков должен быть дизайн типичного приложения, это лишь учебный пример, иллюстрирующий работу механизма всплывающих уведомлений.

Первый этап создания всплывающего уведомления подразумевает включение соответствующего режима в приложении. Для того чтобы это сделать, откройте файл Package.appmanifest, перейдите на вкладку Application UI (Интерфейс приложения) редактора манифеста и в раскрывающемся списке Toast Capable (Всплывающие уведомления) выберите вариант Yes (Да), как показано на рис. 7.9.

После включения режима вывода всплывающих уведомлений можно приступать к их созданию. В нашем учебном приложении используются расширения, взятые из примеров кода для Windows 8, о которых мы говорили

ранее в этой главе. Для уведомлений предусмотрено восемь шаблонов. В табл. 7.1 приведен их перечень и даны описания.

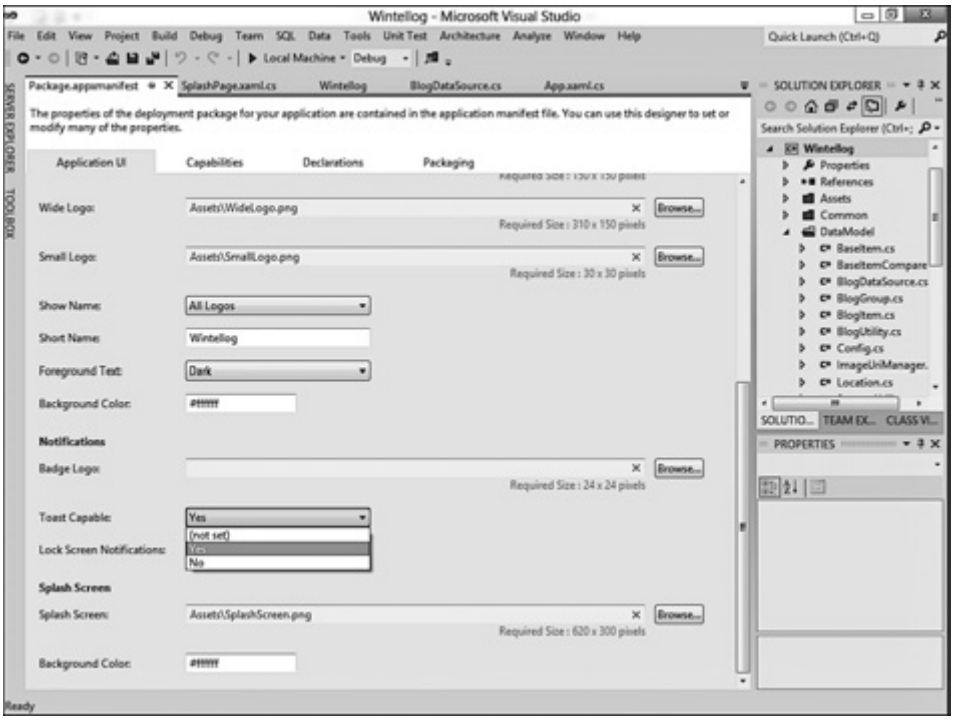


Рис. 7.9. Включение режима вывода всплывающих уведомлений

Таблица 7.1. Всплывающие уведомления

Шаблон	Описание
ToastText01	Текст, который может занимать до трех строк
ToastText02	Строка текста, выделенная полужирным шрифтом, за которой следует до двух строк обычного текста
ToastText03	Текст, выделенный полужирным шрифтом, который может занимать до двух строк, и одна строка обычного текста
ToastText04	Одна строка полужирного текста, за которым следуют две строки обычного текста
ToastImageAndText01	Большое изображение и текст, который может занимать до трех строк

Шаблон	Описание
ToastImageAndText02	Большое изображение, одна строка полужирного текста и до двух строк обычного текста
ToastImageAndText03	Большое изображение, до двух строк полужирного текста и одна строка обычного текста
ToastImageAndText04	Большое изображение, одна строка полужирного текста и две строки обычного текста

В файле вспомогательного кода SplashPage.xaml.cs вы можете увидеть программный код создания всплывающего уведомления. Хотя подобный код, вероятнее всего, будет исполняться на сервере, в нашем случае он включен в приложение с учебными целями. Код создает шаблон и задает значения, которые используются в этом шаблоне:

```
var notificationTemplate =
  ToastContentFactory.CreateToastImageAndText02();
notificationTemplate.TextHeading.Text = item.Group.Title;
notificationTemplate.TextBodyWrap.Text = item.Title;
notificationTemplate.Image.Src = item.DefaultImageUri.ToString();
notificationTemplate.Image.Alt = item.Title;
```

Уведомления похожи на дополнительные плитки, так как они тоже могут передавать приложению аргументы, имеющие отношение к контексту приложения. Свойство `Launch` служит для задания таких аргументов. В учебном приложении для уведомлений используется тот же формат, что и для дополнительных плиток:

```
notificationTemplate.Launch = string.Format("Item={0}", item.Id);
```

При обработке данных параметров мы можем воспользоваться готовым программным кодом. Если вы хотите, чтобы можно было отличать обычный запуск приложения от запуска, выполненного с помощью дополнительной плитки или всплывающего уведомления, можете добавить в аргумент дополнительные данные. Класс `ToastNotificationManager` предоставляет функциональность, необходимую для планирования всплывающих уведомлений. Он позволяет создать экземпляр объекта уведомления, к которому можно обратиться, чтобы узнать, включен ли режим всплывающих уведомлений:

```
var notifier = ToastNotificationManager.CreateToastNotifier();
if (notifier.Setting == NotificationSetting.Enabled) { ... }
```

Если вы хотите знать, почему отключен режим уведомлений, можете обратиться к параметру `Setting`. Он может содержать следующие значения перечисления `NotificationSetting`:

- `Enabled`. Режим всплывающих уведомлений включен, с ними можно работать.
- `DisabledByGroupPolicy`. Администратор отключил все уведомления посредством групповой политики.
- `DisabledByManifest`. При создании приложения не объявлено о намерении пользоваться всплывающими уведомлениями, то есть в манифесте приложения не был выбран вариант Yes (Да) в раскрывающемся списке Toast Capable (Всплывающие уведомления).
- `DisabledForApplication`. Пользователь явно отключил режим уведомлений для данного приложения.
- `DisabledForUser`. Пользователь или администратор отключил режим уведомлений на устройстве пользователя.

Когда вы выяснили, что работать с уведомлениями можно, и создали шаблон, можете просто создать уведомление из шаблона и вывести его на экран:

```
var toast = notificationTemplate.CreateNotification();
notifier.Show(toast);
```

Если вместо немедленного показа уведомления вы хотите запланировать его показ в будущем (именно так сделано в учебном приложении), просто укажите время вывода уведомления и создайте объект типа `ScheduledToastNotification`. Вместо вызова метода `Show` нужно передать экземпляр этого объекта методу `AddToShedule`:

```
var toast = notificationTemplate.CreateNotification();
var date = DateTimeOffset.Now.AddSeconds(30);
var scheduledToast = new ScheduledToastNotification(toast.Content,
date);
notifier.AddToSchedule(scheduledToast);
```

Для того чтобы протестировать всплывающие уведомления, можно открыть локальное хранилище данных приложения и удалить один-два загруженных ранее поста. Запустите приложение. Когда оно выполнит инициализацию и выведет на экран первую страницу, закройте его (выполнив жест скольжения от верхнего края к нижнему или нажав клавиши Alt+F4). Подождите около 30 секунд, и вы увидите всплывающее уведомление. Можете либо закрыть его, либо щелкнуть на нем. Так как уведомление

настраивалось для показа конкретного поста, щелчок на уведомлении приведет к запуску приложения и выводу на экран этого поста. На рис. 7.10 показано несколько уведомлений, появляющихся поверх погодного приложения.

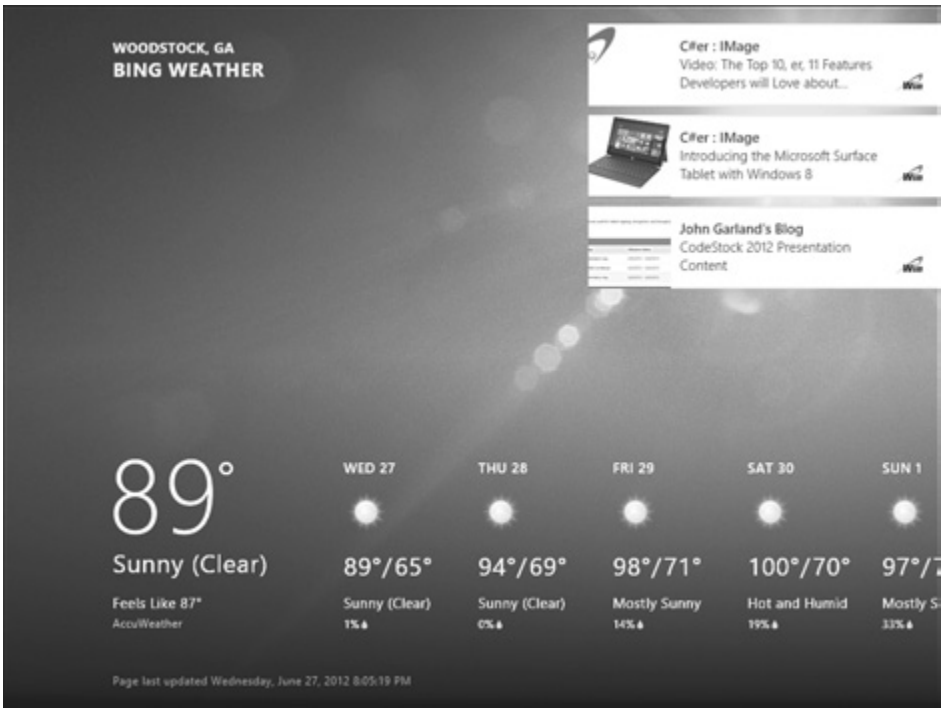


Рис. 7.10. Всплывающие уведомления располагаются поверх окна погодного приложения

Для того чтобы закрыть уведомление, можете либо перетащить его за пределы экрана, либо навести указатель мыши на его верхний правый угол и воспользоваться появившейся там кнопкой закрытия, либо дождаться, когда оно исчезнет само. Касание уведомления или щелчок на нем приведут к запуску приложения.

Кроме того, вы можете изменить длительность звучания, сопровождающего уведомление, и само это звучание. Для этого нужно задать свойство `Audio.Content` в шаблоне. В частности, доступны звуки мгновенного сообщения, оповещения, вызова, прихода электронного письма, напоминания, SMS-сообщения. Звук можно вообще отключить. Установка свойства `Duration` в значение `ToastDuration.Long` позволяет продлить показ уведомления до 25 секунд вместо стандартного периода в 7 секунд.

СОВЕТ

Если приложение не исполняется, всплывающее уведомление инициирует запуск приложения и передает ему заданные аргументы. Также есть возможность отреагировать на касание пользователем уведомления и в том случае, если приложение при появлении уведомления уже исполняется. Для этого нужно перехватить событие Activated экземпляра уведомления при вызове метода CreateNotification. Когда уведомление закрывается либо по инициативе пользователя, либо по истечении тайм-аута, отреагировать на это можно, перехватив событие Dismissed. Проанализировав аргументы этого события, можно узнать причину, по которой было закрыто уведомление.

Плитки и всплывающие уведомления, представленные во всех предыдущих примерах, выводились самим приложением. Это достаточно необычно, так как чаще приходится обновлять приложение, когда оно не исполняется. Погодное приложение может обновлять плитки при изменении погоды и в особых случаях показывать уведомления. Как сделать так, чтобы приложение оставалось подключенным и активным, когда оно не исполняется? Ответ на этот вопрос кроется в push-уведомлениях.

Сервис уведомлений Windows

Сервис уведомлений Windows (Windows Notification Service, WNS) позволяет доставлять данные для плиток и всплывающих уведомлений через Интернет. Именно с его помощью можно поддерживать актуальность данных приложения даже тогда, когда оно не запущено. Этот сервис поддерживает взаимодействие с приложениями и даже позволяет разрешать непростые ситуации, связанные с доставкой данных приложениям, защищенным брандмауэром. Он создан для поддержки миллионов пользователей и, что самое приятное, совершенно бесплатен.

Для применения WNS-сервиса требуется пройти пять шагов. Эти шаги одинаковы независимо от того, что требуется обновлять — плитки, всплывающие уведомления или простые индикаторы. Графически эти шаги иллюстрирует рис. 7.11.

В частности, речь идет о следующих шагах:

1. Приложение запрашивает канал. Это уникальный URI-идентификатор, который используется для взаимодействия с конкретным приложением, установленным на конкретному устройстве, работающем под управлением Windows 8.

Рис. 7.11. Пошаговая процедура использования сервиса уведомлений Windows

2. WNS назначает канал и отправляет эту информацию обратно приложению.

3. Теперь приложение должно сообщить об этом канале серверу, управляющему его уведомлениями. Обычно это веб-сервер, который создаете вы или ваша организация для рассылки подобных уведомлений и управления каналами. Часто URI-идентификатор канала отправляется вместе с уникальным идентификатором устройства, в результате уведомления возвращаются к исходному устройству.

4. Когда серверу нужно отправить уведомление, он передает контент уведомления WNS-сервису, используя полученный URI-идентификатор канала.

5. Устройство, работающее под управлением Windows 8, поддерживает связь с WNS-сервисом даже тогда, когда между устройством и WNS-сервисом находится брандмауэр. Когда данные обновления поступают на устройство, выполняется обновление индикатора или плитки либо инициируется вывод всплывающего уведомления.

Действующим подключением управляет приложение, но период функционирования подключения может истечь. Поэтому рекомендуется устанавливать и сохранять подключение при каждом запуске или возобновлении

работы приложения. Это позволит гарантировать, что у сервиса будут самые свежие данные.

В дополнение к масштабированию, которое требуется для взаимодействия с огромным количеством устройств, WNS-сервис должен решать вопросы безопасности. Хотя любое приложение для Windows 8 может запросить у WNS-сервиса канал обновлений, для отправки сообщений нужна авторизация, подразумевающая использование особого ключа. Этот ключ уникален и основан на комбинации уникального идентификатора безопасности приложения и создаваемого вами общего секретного кода.

Для того чтобы получить учетные данные, необходимые для взаимодействия с WNS-сервисом, можно пойти одним из двух путей. Предпочтительно воспользоваться возможностями учетной записи разработчика Магазина Windows. Подробнее о центре разработчиков вы можете узнать на странице http://msdn.microsoft.com/ru-ru/windows/apps/.

Для того чтобы получить доступ к учетной записи разработчика или создать новую, щелкните на ссылке Dashboard (Информационная панель) и следуйте указаниям системы. На момент написания этих строк Магазин Windows был еще закрыт для регистрации, к тому же последовательность действий, необходимая для регистрации учетной записи, может измениться после выпуска финального релиза Windows 8. После того как вы создадите учетную запись разработчика, чтобы получить необходимые учетные данные, можете воспользоваться инструкциями со страницы http://msdn.microsoft.com/ru-ru/library/windows/apps/hh465407.aspx.

Если у вас нет учетной записи разработчика или вы хотите получить временный доступ к сервису для тестирования, можете воспользоваться сайтом управления приложениями Windows Live: http://go.microsoft.com/fwlink/?linkid=227235.

Для доступа к нему нужна учетная запись Windows Live. После того как вы авторизуетесь на сайте, можете ввести сведения о приложении из его манифеста. В частности, вам понадобятся данные из полей Package Display Name (Отображаемое имя пакета) и Publisher (Издатель) на вкладке Packaging (Пакетирование) манифеста приложения. На рис. 7.12 показаны эти данные для приложения Wintellog.

Отображаемое имя пакета в данном случае — `Wintellog`, издатель — `CN=Jeremy` (не путать с отображаемым именем издателя — `Jeremy`). Введите данные значения в соответствующие поля. Когда вы примете условия оказания услуг и отправите форму, то увидите страницу, похожую на страницу с рис. 7.13.

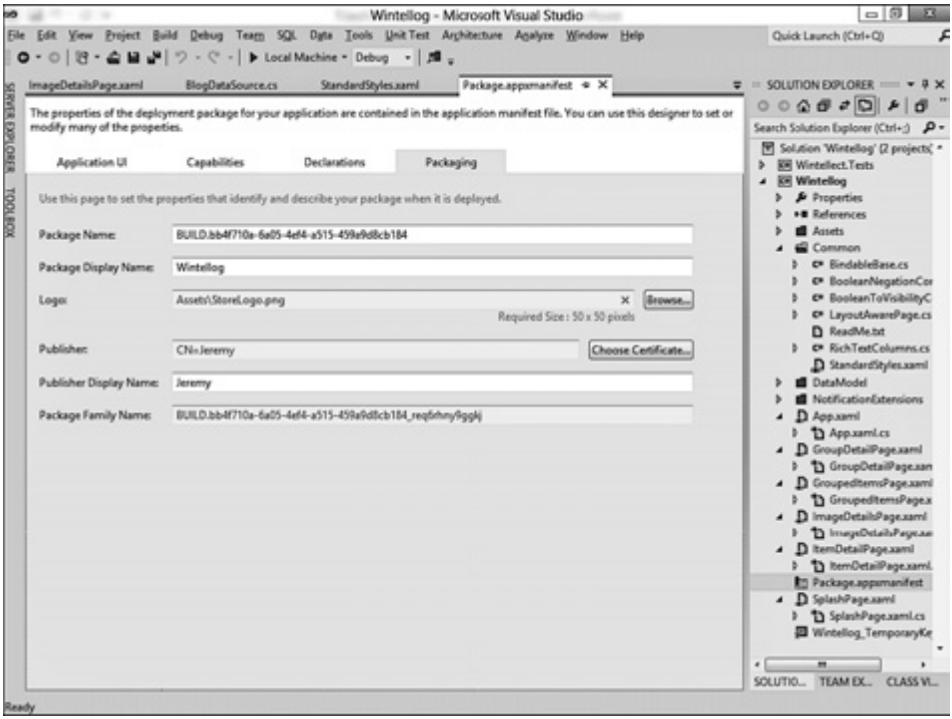


Рис. 7.12. Сведения о пакете приложения

Вот ключевые данные, которые нас интересуют:

- Используя предоставленное имя пакета (Package Name), нужно обновить соответствующий параметр в манифесте приложения. Если имя пакета вашего приложения уже задано, просто замените его тем, которое предоставлено здесь.
- Секретный код клиента (Client Secret) — это специальный код, который вы будете использовать для авторизации своих push-уведомлений. Он должен храниться на вашем сервере, который занимается отправкой уведомлений.
- Идентификатор безопасности пакета (Package Security Identifier) уникален для каждого пакета и также используется для авторизации. Секретный код и идентификатор потребуются для проверки пакета при отправке запроса на получение push-уведомлений.

Когда у вас есть вся необходимая информация, можете начинать отправку push-уведомлений. Первый шаг заключается в запросе канала для уведомлений. Наше приложение делает это при запуске и сохраняет сведения

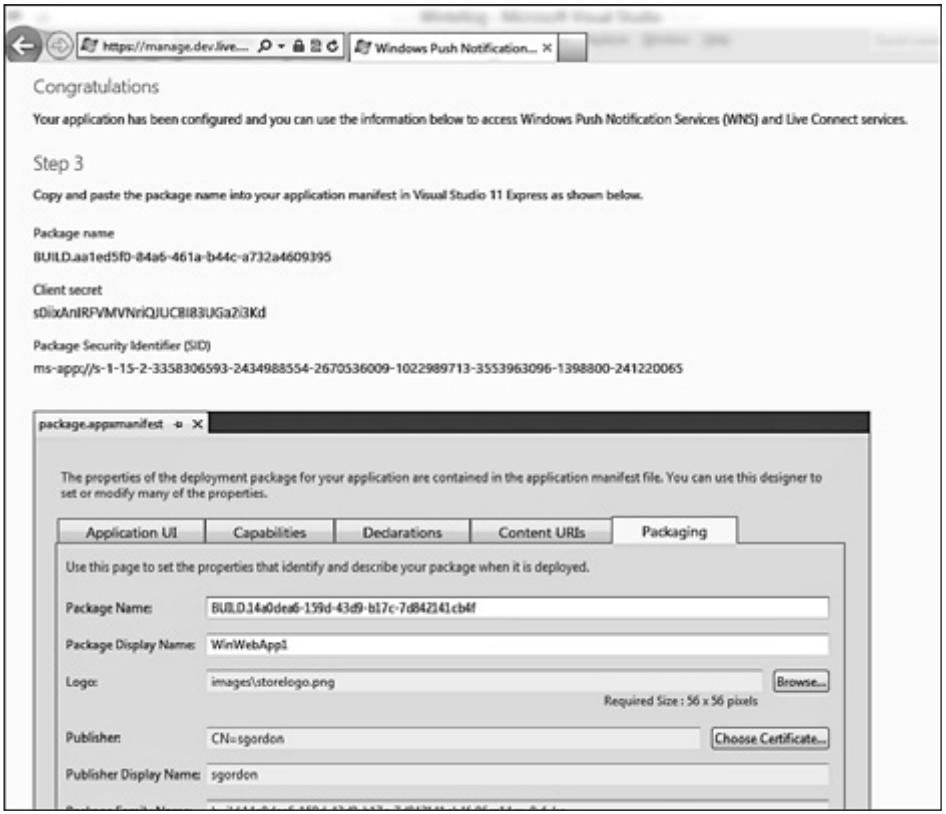


Рис. 7.13. Учетные данные для доступа к WNS-сервису

о канале для дальнейшего использования (этот код находится в файле App.xaml.cs):

```
Channel = await PushNotificationChannelManager
  .CreatePushNotificationChannelForApplicationAsync();
```

Данный вызов предельно прост и понятен. Если при его выполнении будет вброшено исключение, приложение сохранит это исключение. После создания канала его обычно отправляют на сервер поддержки уведомлений, который сохраняет его и использует для последующего взаимодействия с клиентским приложением. Я не собираюсь предлагать вам создавать веб-сервис и обеспечивать его хостинг, поэтому в данном примере мы просто сохраняем URI-идентификатор канала, который вы можете скопировать и использовать во вспомогательном приложении. Далее в этой главе мы рассмотрим простейший способ организации веб-сайта для регистрации уведомлений.

С главной страницы приложений Wintellog (`GroupedItemsPage`) вы можете открыть панель приложения, на которой расположена кнопка Show Channel (Отобразить канал). Когда вы нажимаете на эту кнопку, URI-идентификатор канала копируется в буфер обмена. Данная операция выполняется с помощью объекта `DataPackage`. Этот же класс обеспечивает передачу данных с помощью контракта Share Contract (Общий доступ), но здесь он применяется для работы с буфером обмена:

```
var dataPackage = new DataPackage();
dataPackage.SetText(App.Instance.Channel.Uri);
Clipboard.SetContent(dataPackage);
```

Вы увидите диалоговое окно, которое сообщит, что копирование успешно завершено, либо уведомит об ошибке, если приложение не получит данные канала. Теперь, когда у вас есть URI-идентификатор канала, можете скомпилировать и запустить приложение PushNotificationTest. Это приложение имитирует действия веб-службы, отправляющей уведомления приложению Wintellog2. Главная страница этого приложения показана на рис. 7.14.

Рис. 7.14. Приложение для тестирования push-уведомлений

Поместите курсор в поле Paste Channel Here (Вставьте сюда URI канала) и вставьте туда URI-идентификатор канала, который был помещен в буфер обмена приложением Wintellog2. Теперь вы можете отправлять уведомления для этого приложения. Например, переместите ползунок в любую позицию и нажмите кнопку Badge Notification (Уведомление для индикатора событий). Если вы увидите диалоговое окно, сообщающее об успешном проведении операции, вернитесь на начальный экран и взгляните на плитку приложения Wintellog2. На плитке должно появиться число, которое вы задали перед отправкой push-уведомления. Также вы можете ввести любой текст и использовать его либо для обновления плитки, либо для вывода уведомления.

Программный код приложения PushNotificationTest демонстрирует, что должна делать веб-служба, чтобы отправлять push-уведомления. Схема на рис. 7.11 слегка упрощена, так как прежде, чем отправлять уведомления, нужно пройти авторизацию. При авторизации используется протокол OAuth, в ходе этой операции получают специальный код, который применяется при отправке уведомлений. OAuth — это популярный открытый протокол для безопасной авторизации; детали вы можете найти на странице http://oauth.net/.

Приложение определяет контракт, который используется для взаимодействия с сервисом:

```
[DataContract]
public class OAuthToken
{
  [DataMember(Name = "access_token")]
  public string AccessToken { get; set; }
  [DataMember(Name = "token_type")]
  public string TokenType { get; set; }
}
```

Именно здесь используются секретный ключ и идентификатор безопасности, которые вы получили ранее (мои заданы в коде приложения). Первый шаг заключается в создании строки, которая нужна для того, чтобы сервис предоставил код доступа:

```
var body =
    String.Format("grant_type=client_credentials&client_id={0}&
    ↳ client_secret={1}&scope=notify.windows.com", urlEncodedSid,
    ↳ urlEncodedSecret);
```

Запрос всегда выглядит одинаково, за исключением идентификатора клиента и секретного ключа. Они из необработанного исходного формата преобразованы в формат, подходящий для передачи через Интернет. Например, секретный ключ закодирован следующим образом:

```
const string SECRET = "4n3ZeAodQpNmAQiBIpl2Pb6gcyKetITN";
var urlEncodedSecret = WebUtility.UrlEncode(SECRET);
```

Ваше приложение, вероятнее всего, будет содержать эти данные в зашифрованном файле конфигурации или в базе данных и при необходимости использовать их. Сообщение форматируется с помощью специального класса, который помогает упаковывать сообщения для отправки по протоколу

HTTP. В данном случае контент состоит из строковых данных, поэтому применяется вспомогательный класс `StringContent`:

```
var httpBody = new StringContent(body,
Encoding.UTF8, "application/x-www-form-urlencoded");
```

Затем для отправки запроса используется экземпляр `HttpClient`. В результате мы должны получить маркер, который содержит код доступа для отправки push-уведомлений:

```
var client = new HttpClient();
var response = await client.PostAsync(
  new Uri("https://login.live.com/accesstoken.srf", UriKind.
    Absolute),
    httpBody);
var oAuthToken = GetOAuthTokenFromJson(
  await response.Content.ReadAsStringAsync());
return oAuthToken.AccessToken;
```

Запрос маркера всегда осуществляется с ресурса https://login.live.com/accesstoken.srf путем отправки секретного ключа и идентификатора клиента. Когда маркер получен, можно извлечь из него код доступа и использовать его для отправки push-уведомлений. Хорошо то, что в этих уведомлениях используется тот же XML-код, который мы ранее применяли для локального обновления плиток и индикаторов, а также для отправки уведомлений. Единственная разница заключается в том, что данный XML-код направляют в специальный канал, полученный приложением.

Например, в коде отправки обновления для индикатора используются те же самые классы расширения, предоставляемые SDK для отправки этого обновления облачному сервису:

```
var badge = new BadgeNumericNotificationContent(
  (uint)BadgeValue.Value);
await PostToCloud(badge.CreateNotification().Content, "wns/badge");
```

Единственная разница между этим кодом и вызовом, который генерирует уведомление, заключается в его контенте и типе:

```
var tile = TileContentFactory.CreateTileSquareText02();
tile.TextHeading.Text = title;
tile.TextBodyWrap.Text = text;
await PostToCloud(bigTile.CreateNotification().Content);
```

При вызове `PostToCloud` производится извлечение XML-данных из уведомления, и сообщение подготавливается для передачи WNS-сервису:

```
var content = xml.GetXml();
var requestMsg = new HttpRequestMessage(HttpMethod.Post, uri);
requestMsg.Content =
  new ByteArrayContent(Encoding.UTF8.GetBytes(content));
```

Далее производится установка заголовков. Сам по себе пакет — это данные в формате XML, однако тип конкретного пакета отображается на `wns/badge`, `wns/tile` или `wns/notification` в зависимости от того, что именно вы хотите отправить в push-уведомлении. Ранее полученный код доступа размещается в заголовке аутентификации:

```
requestMsg.Content.Headers.ContentType =
  new MediaTypeHeaderValue("text/xml");
requestMsg.Headers.Add("X-WNS-Type", type);
requestMsg.Headers.Authorization =
  new AuthenticationHeaderValue("Bearer", _accessCode);
```

И наконец, остается отправить сообщение. Ответ сервиса сохраняется, чтобы в случае неудачи можно было узнать подробности:

```
var responseMsg = await new HttpClient().SendAsync(requestMsg);
message = string.Format("{0}: {1}", responseMsg.StatusCode,
  await responseMsg.Content.ReadAsStringAsync());
```

Вот и все, что нужно сделать! Тот же самый код можно легко исполнить на веб-сервере. Вы можете использовать исходный код расширений для всплывающих уведомлений и плиток, чтобы создать похожие классы на веб-сервере. Когда клиент регистрирует канал, вместо копирования полученного URI-идентификатора в буфер обмена нужно подготовить сообщение для отправки полученных сведений на сервер уведомлений вашего приложения. Это отличный пример сценария, для реализации которого идеально подходит платформа Azure, позволяющая без проблем настроить все необходимое для работы с уведомлениями. Узнать подробности об Azure и о том, как эта платформа работает с приложениями для Windows 8, можно, воспользовавшись материалами проекта Windows Azure Toolkit для Windows 8, доступного на странице http://watwindows8.codeplex.com/.

Чтобы вам проще было создавать всплывающие уведомления, можете беспрепятственно использовать вспомогательные методы для выполнения аутентификации или любой другой код из этого примера в собственных приложениях.

Выводы

В этой главе вы узнали, как сделать так, чтобы ваши приложения оставались активными, обновленными и подключенными даже тогда, когда они не запущены. С помощью индикаторов вы можете информировать пользователя об ожидающей его новой информации. Плитки могут динамически обновляться, позволяя предварительно знакомиться с изображениями и контентом приложения, а уведомления представляют собой своеобразные «плакаты», перекрывающие текущее приложение, с которым работает пользователь, и дающие ему возможность отреагировать на уведомление, запустив с его помощью ваше приложение. Всеми этими средствами можно пользоваться как из кода приложения, так и через внешний сервер посредством сервиса уведомлений Windows (WNS).

В следующей главе вы узнаете, как придать вашим приложениям особое очарование с помощью чудо-кнопок и контрактов. Каждый раз, когда вы запускаете или отлаживаете приложение, вы уже используете контракт запуска приложения (Launch). Вы сталкивались и с некоторыми другими контрактами, в том числе с контрактами Settings (Параметры) и Share (Общий доступ). В следующей главе вы узнаете подробности об этих контрактах, а также о том, как интегрировать в ваши приложения контракт Search (Поиск), и даже о том, как работать с потоковыми мультимедийными данными с помощью контракта PlayTo (Воспроизведение на устройстве).

Чудо-кнопки для вашего приложения

8

Представьте, что вы пытаетесь создать приложение настолько изощренное, что способно предугадывать, как взаимодействовать со всеми программами, которые еще только будут написаны для данной платформы. Это дало бы потрясающую возможность интеграции с приложениями, которые сегодня еще даже не созданы. При этом пользователь смог бы легко настраивать приложения, передавать между ними информацию о контактах или изображения. Вам, как разработчику, пришлось бы, конечно, проанализировать все программные продукты, которые могут появиться, чтобы подготовить соответствующие интерфейсы и адаптеры, позволяющие вашему приложению находить с ними общий язык.

В Windows 8 столь непростая задача возложена на операционную систему. Это позволяет разработчику не задумываться о том, как приложения будут взаимодействовать друг с другом. Вместо того чтобы заниматься изучением особенностей интерфейсов других приложений, разработчику достаточно разобраться со стандартным набором соглашений, применимых ко всем программам для Windows 8. Эти соглашения называются *контрактами* (contracts). Контракты предоставляют вашему приложению возможность взаимодействовать с другими приложениями, не задумываясь об особенностях их внутренних механизмов для работы с данными. Эта чрезвычайно мощная возможность впервые появилась в Windows 8.

Контракт в Windows 8 — это соглашение между приложениями. Он определяет набор требований, которым должна соответствовать программа, чтобы принимать участие в некоторой схеме взаимодействия с другими продуктами. Кроме того, Windows 8 поддерживает расширения (extensions), которые представляют собой соглашения между приложениями и операционной системой. Например, когда Windows 8 предлагает пользователю выбрать фото для его учетной записи, ваше приложение может появиться в списке доступных пользователю программ, если вы решите оснастить свое приложение расширением для обеспечения подобной функциональности.

Windows 8 поддерживает шесть контрактов. Их названия и краткие описания приведены в табл. 8.1. Далее в этой главе вы узнаете, как реализовать многие из них.

Таблица 8.1. Контракты в Windows 8

Название контракта	Описание
File Picker (Средство выбора файлов)	Пользователи, работающие с одними приложениями, получают возможность брать файлы напрямую из других приложений. http://msdn.microsoft.com/ru-ru/library/windows/apps/hh465174.aspx
Cached File Updater (Средство обновления кэшированных файлов)	Позволяет приложению осуществлять обновление определенных файлов, в итоге пользователь может работать с таким приложением как с централизованным хранилищем данных. Например, так может производиться синхронизация файлов, хранящихся локально, с файлами, расположенными в облачном сервисе. http://msdn.microsoft.com/ru-ru/library/windows/apps/hh465192.aspx
PlayTo (Воспроизведение на устройстве)	Поддерживает потоковое вещание цифровых мультимедийных данных на DLNA-устройствах, таких как проекторы и телевизоры. Для того чтобы узнать подробности о стандарте DLNA (Digital Living Network Alliance), обратитесь к ресурсу http://www.dnla.org/. http://msdn.microsoft.com/ru-ru/library/windows/apps/hh465176.aspx
Search (Поиск)	Обеспечивает интеграцию приложения в единую среду поиска данных, давая пользователю возможность осуществлять поиск в приложении из других мест системы и передавать результаты поиска из приложения другим поставщикам данных поиска. Например, файловой системе. http://msdn.microsoft.com/ru-ru/library/windows/apps/hh465231.aspx

продолжение ⇩

Таблица 8.1 *(продолжение)*

Название контракта	Описание
Settings (Параметры)	Предоставляет стандартное место для размещения параметров конкретного приложения, с которым может работать пользователь. http://msdn.microsoft.com/ru-ru/library/windows/apps/hh770540.aspx
Share (Общий доступ)	Дает пользователям возможность передавать данные из вашего приложения в другие и позволяет приложению принимать данные, предоставленные для общего доступа. Среди таких данных может быть, например, текст, гиперссылки, файлы, изображения. http://msdn.microsoft.com/ru-ru/library/windows/apps/hh758314.aspx

Приложения для Windows 8 могут также использовать расширения. Таких расширений десять. Они перечислены в табл. 8.2. Расширения представляют собой соглашения между приложением и Windows 8.

Таблица 8.2. Расширения для Windows 8

Расширение	Описание
Account Picture Provider (Поставщик аватаров)	Реализация приложением данного расширения указывает на то, что оно может принимать или предоставлять изображения, которые пользователи могут указывать в качестве изображений для своих учетных записей. http://go.microsoft.com/fwlink/?LinkId=231579
AutoPlay (Автозапуск приложения)	Позволяет выводить приложение в списке доступных приложений при подключении к компьютеру внешнего устройства. Если приложение будет выбрано в списке, оно может автоматически запускаться при последующих подключениях этого устройства. http://msdn.microsoft.com/ru-ru/library/windows/apps/hh452731.aspx
Background Tasks (Фоновые задачи)	Используется приложениями для запуска программного кода в условиях, когда приложение приостановлено или остановлено. Фоновые задачи предназначены для решения небольших задач, не требующих участия пользователя. http://www.microsoft.com/en-us/download/details.aspx?id=27411

Расширение	Описание
Camera Settings (Параметры камеры)	Указывает на то, что приложение может предоставить пользовательский интерфейс для настройки камеры и выбора эффектов, когда камера служит для того, чтобы сделать фотоснимок или видеозапись. http://msdn.microsoft.com/library/windows/hardware/hh454870
Contact Picker (Окно выбора контактов)	Дает приложению возможность предоставлять контактные сведения из внутренней адресной книги. Приложения, реализующие это расширение, выводятся в списке, который предоставляется пользователю, когда ему нужна подобная информация. http://msdn.microsoft.com/ru-ru/library/windows/apps/hh464939.aspx
File Activation (Активация файла)	Позволяет приложению обрабатывать файлы существующих типов, а также и устанавливать новые типы файлов, для обработки которых может быть зарегистрировано приложение. http://msdn.microsoft.com/ru-ru/library/windows/apps/hh452684.aspx
Game Explorer (Проводник игр)	Реализация данного расширения указывает на то, что приложение является игрой. Это позволяет Windows 8 ограничивать доступ к игре в том случае, если пользователь активизирует параметры семейной безопасности. http://msdn.microsoft.com/ru-ru/library/windows/apps/hh465153.aspx
Print Task Settings (Параметры задачи печати)	Дает приложению возможность предоставлять интерфейс для настройки параметров печати, которые влияют непосредственно на печатающее устройство. http://msdn.microsoft.com/en-us/library/windows/hardware/br259129
Protocol Activation (Активация протокола)	Приложения получают возможность обрабатывать как существующие, так и новые протоколы для различных целей. http://msdn.microsoft.com/ru-ru/library/windows/apps/hh452686.aspx
SSL Certificates (SSL-сертификаты)	Предоставляет средства для установки цифровых сертификатов с помощью приложения. http://msdn.microsoft.com/ru-ru/library/windows/apps/hh465012.aspx

К настоящему моменту мы уже знакомы с панелью чудо-кнопок Windows 8, которую можно вызвать, выполнив жест скольжения от правого края экрана или нажав сочетание клавиш Win+C. Каждая из *чудо-кнопок* (charms) связана с определенным контрактом. Чудо-кнопка Start (Пуск) уникальна, так как она открывает начальный экран и позволяет активизировать приложения, запуская их с помощью плиток. Чудо-кнопка Device (Устройства) активизирует контракт воспроизведения на устройстве (PlayTo) и показывает доступные целевые DLNA-устройства. Чудо-кнопки Search (Поиск), Share (Общий доступ) и Settings (Параметры) связаны с одноименными контрактами. Именно ими мы будем заниматься в этой главе.

Поиск

Кнопка Start (Пуск) была в центре внимания, когда Билл Гейтс почти 20 лет назад анонсировал Windows 95. Эта кнопка стала почти синонимом Windows, поэтому для многих стало неожиданностью, когда выяснилось, что в Windows 8 ее не будет. Во многих статьях и постах делались попытки выяснить причины, по которым эту кнопку убрали и заменили ее сочетанием нового начального экрана с мощным механизмом поиска. Microsoft сообщает, что основная причина заключается в том, что почти 70 % всех операций поиска в операционной системе производятся для того, чтобы находить и запускать программы (см. http://blogs.msdn.com/b/b8_ru/archive/2011/10/25/171-187-designing-search.aspx).

В Windows 8 механизм поиска является центральным. Я использую Windows 8 как основную операционную систему с того момента, как она впервые была представлена в редакции Developer Preview в сентябре 2011 года. Хотя мне понадобилось некоторое время, чтобы избавиться от условного рефлекса, заставляющего меня тянуться к кнопке Start (Пуск), чтобы запустить программу, в конце концов новые инструменты Windows мне понравились. Я могу запустить любую из сотен или тысяч программ, нажав всего несколько клавиш.

Для того чтобы понять, как все это работает, давайте для тренировки запустим приложение Блокнот (Notepad). Щелкните на плитке Desktop (Рабочий стол) на начальном экране, затем нажмите клавишу Windows, после чего начните вводить буквы Б и Л (N и O в латинской раскладке) и нажмите клавишу Enter. Всего четыре нажатия клавиш. Весьма вероятно, если только у вас не установлены другие программы с похожими именами, что указанная последовательность действий приведет к запуску Блокнота. Так происходит потому, что начальный экран (на который мы перешли, нажав клавишу Windows) по умолчанию настроен на поиск приложений. Если в ходе

поиска было бы обнаружено лишь частичное совпадение, приложения, которыми пользуются чаще, чем другими, оказались бы на первых позициях результатов поиска. На рис. 8.1 показан результат нажатия клавиши вызова начального экрана и ввода двух символов. Обратите внимание, что в списке результатов представлены все приложения, в именах которых есть сочетание «no», при этом приложение Notepad (Блокнот) выделено в качестве приложения, запускаемого по умолчанию (то есть по нажатию клавиши Enter).

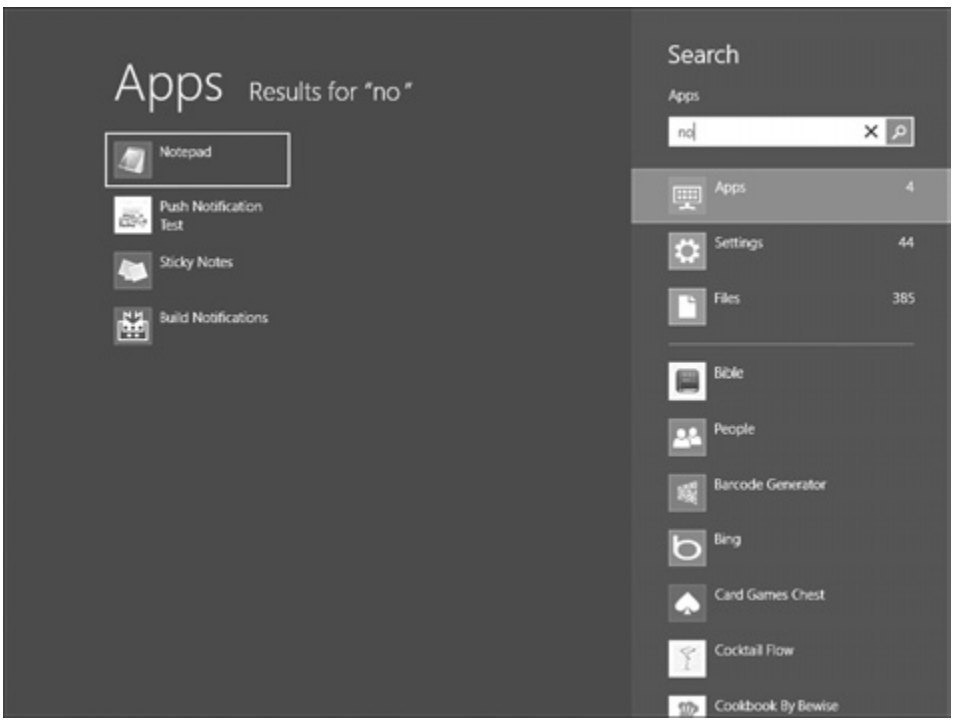


Рис. 8.1. Приложение Notepad (Блокнот) найдено в результате быстрого поиска

Обратите внимание, что доступно три направления поиска — они перечислены в правой верхней части экрана. По умолчанию выполняется поиск приложений, установленных в системе. Вы можете нажать кнопку Settings (Параметры), чтобы увидеть 44 соответствия в параметрах приложений (на вашем компьютере это значение может быть другим), или кнопку Files (Файлы), чтобы по введенному ключевому слову выполнить поиск в файловой системе. Кроме того, вы можете увидеть длинный список приложений, расположенный под этими тремя кнопками. Все эти приложения поддерживают контракт Search (Поиск).

Открыть интерфейс чудо-кнопки поиска можно разными способами. Первый заключается в том, чтобы выполнить жест скольжения от правого края экрана для вызова панели чудо-кнопок, а затем коснуться значка чудо-кнопки поиска. Панель чудо-кнопок можно открыть, нажав сочетание клавиш Win+C. И наконец, можно непосредственно вызвать интерфейс чудо-кнопки поиска, нажав сочетание клавиш Win+Q («Q» в данном сочетании можно воспринимать как сокращение слова «query» — «запрос»). Когда будет открыта панель поиска, наберите слово `chicken` (цыпленок) и коснитесь значка обозревателя Internet Explorer. Поисковый запрос будет автоматически передан веб-браузеру, как показано на рис. 8.2.

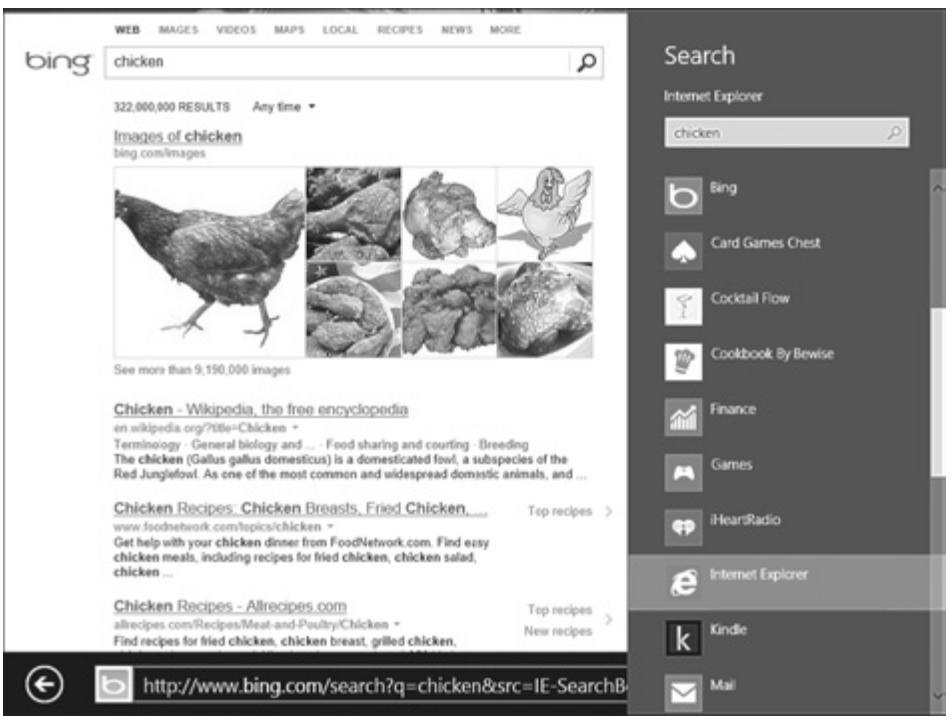


Рис. 8.2. Результаты поиска по запросу «chicken» с помощью Internet Explorer

Не изменяя поисковый запрос, можно выбрать приложение Dictionary.com, если оно у вас установлено, и прочесть статью, посвященную курам (рис. 8.3).

Интеграция поисковых возможностей позволяет производить поиск приложений. Приложения, которые поддерживают контракт поиска, обеспечивают поиск их контента, возможен также поиск других ресурсов. Благодаря этому можно начать поиск в одном приложении и уточнять параметры

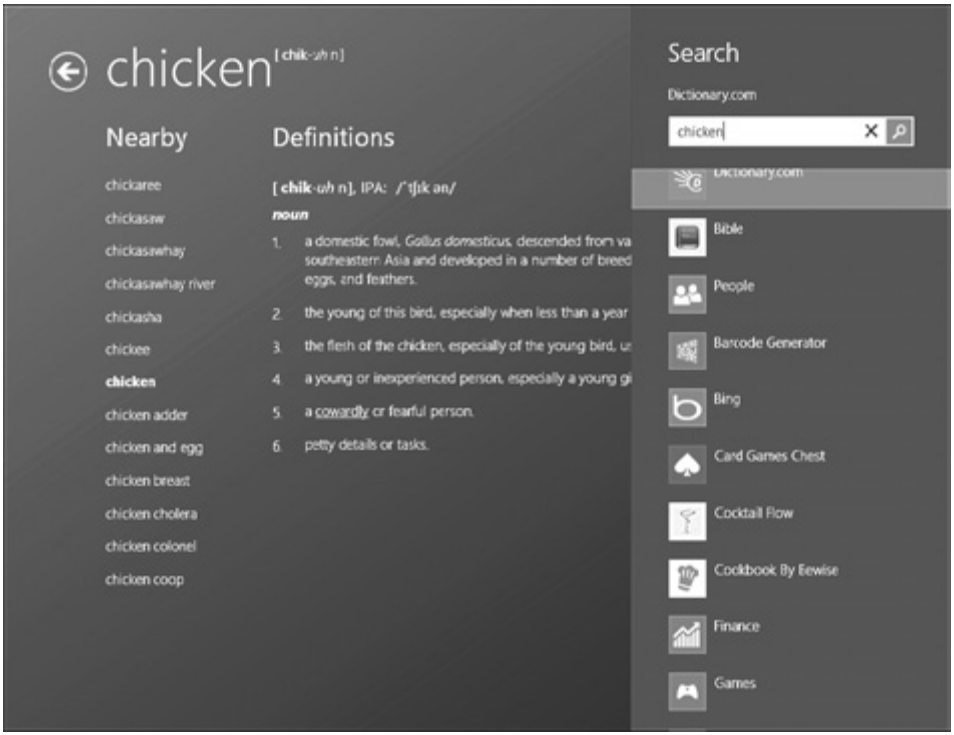


Рис. 8.3. Результаты поиска по запросу «chicken» с помощью Dictionary.com

поиска до тех пор, пока вы не найдете такой источник информации, который способен дать вам то, что вы ищете. Для того чтобы добавить в приложения возможность поиска, используются специальные шаблоны Visual Studio 2012.

В приложении Wintellog3 показан пример интеграции поисковых возможностей. Мой первый шаг в добавлении инструментов поиска заключался в том, что я щелкнул правой кнопкой мыши на значке проекта в обозревателе решений (Solution Explorer), выбрал во всплывающем меню команду Add ▸ New Item (Добавить ▸ Создать элемент) и в появившемся окне выбрал шаблон Search Contract (Контракт поиска), как показано на рис. 8.4.

Добавление шаблона для контракта поиска приводит к нескольким результатам. Поиск включается в список контрактов вашего приложения в раздел объявлений манифеста. Создается базовая XAML-страница для хранения результатов поиска, кроме того, производятся некоторые действия, направленные на то, чтобы приложение могло реагировать на входящие поисковые запросы. Также в ходе этого процесса в класс App добавляется код обработки входящих поисковых запросов. Для большинства приложений

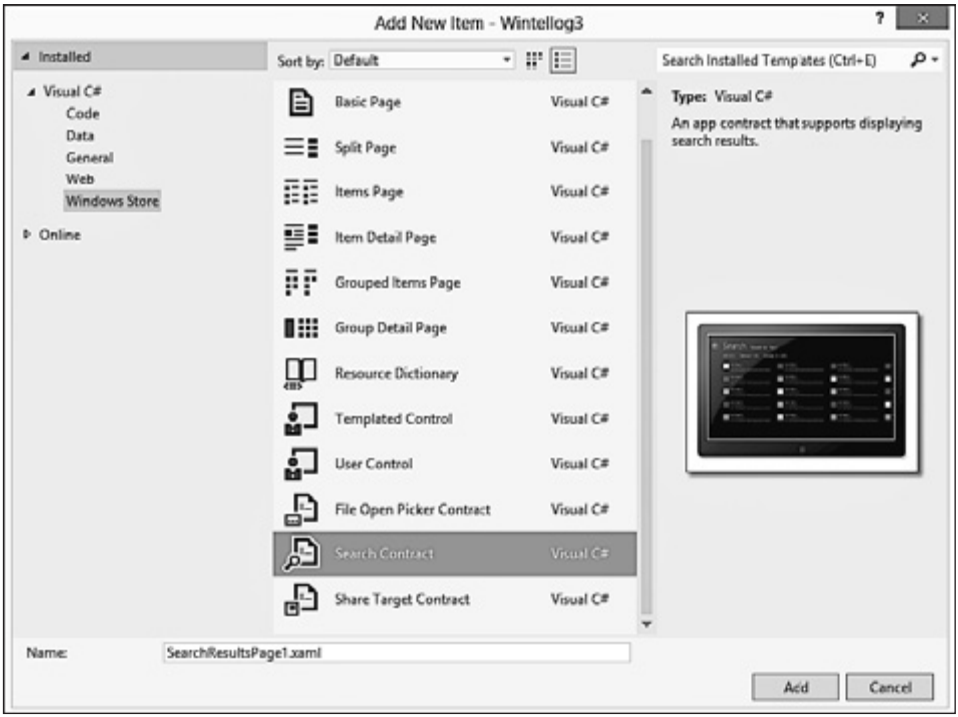


Рис. 8.4. Выбор шаблона для контракта поиска

подобной настройки, выполняемой по умолчанию, вполне достаточно, но так как в приложении Wintellog3 используется расширенная экранная заставка, стандартный программный код был изменен, чтобы перед началом поиска выполнялась загрузка блогов.

В класс `App` я добавил свойство `Extended`, призванное отслеживать, была ли запущена расширенная экранная заставка. Если нет, сначала должна быть загружена страница для получения списка блогов и постов. После загрузки расширенной экранной заставки последующие поисковые запросы направляются сразу на страницу поиска, так как необходимые данные уже загружены. Код для соответствующей проверки размещен в перегруженном методе, который вызывается при инициировании поиска:

```
protected override void
  OnSearchActivated(SearchActivatedEventArgs args)
{
  if (Extended) { SearchResultsPage.Activate(args); }
  else { ExtendedSplash(args.SplashScreen, args); }
}
```

Страница экранной заставки была модифицирована для приема параметра `SearchActivatedEventArgs` в дополнение к уже обрабатываемому ею параметру `LaunchedActivatedEventArgs`. Логика выявления первой загружаемой страницы была дополнена с целью активизировать страницу поиска после передачи аргументов поиска. Это позволило использовать одну и ту же экранную заставку для обработки различных сценариев, охватывающих поиск, первоначальную активизацию, активизацию после завершения и запуск со вспомогательной плитки.

Страница результатов поиска (SearchResultsPage.xaml.cs) содержит все необходимое для реализации поиска. Статический метод `Activate` проверяет, существует ли объект `Frame` (рамка). Если такого объекта нет, он создается и затем в нем выводится страница поиска. Ей передается искомый текст и ссылка на предыдущий контент, что позволяет пользователю вернуться назад после завершения поиска. Этот метод показан в листинге 8.1.

Листинг 8.1. Метод для активизации страницы поиска

```
public static void Activate(SearchActivatedEventArgs args)
{
  var previousContent = Window.Current.Content;
  var frame = previousContent as Frame;
  if (frame != null)
  {
    frame.Navigate(typeof (SearchResultsPage), args.QueryText);
  }
  else
  {
    var page = new SearchResultsPage
    {
      _previousContent = previousContent
    };
    page.LoadState(args.QueryText, null);
    Window.Current.Content = page;
  }
  Window.Current.Activate();
}
```

При загрузке страницы создается набор фильтров. Фильтры позволяют сужать область поиска. Вы можете самостоятельно решить, какие именно фильтры нужны вашему приложению. В приложении Wintellog3 пользователь может сузить область поиска до отдельного блога. Сначала вычисляется общее количество найденных соответствий:

```
var total = (from g in App.Instance.DataSource.GroupList
             from i in g.Items
             where i.Title.ToLower().Contains(query) ||
               i.Description.ToLower().Contains(query)
             select i).Count();
```

Затем создается список фильтров с записью для каждого из найденных соответствий:

```
var filterList = new List<Filter>
{
  new Filter(string.Empty, "All", total, true)
};
```

Класс `Filter` содержит идентификатор для каждого блога, его заголовок и количество соответствий поисковому запросу:

```
filterList.AddRange(from blogGroup in
  App.Instance.DataSource.GroupList
    let count = blogGroup.Items.Count(
      i => i.Title.ToLower().Contains(query) ||
      i.Description.ToLower().Contains(query))
      where count > 0
      select new Filter(blogGroup.Id, blogGroup.Title, count));
```

Фильтры и текст запроса сохраняются в базовой странице, при создании страницы используется механизм привязки данных. По умолчанию выводятся все найденные результаты. Каждый раз, когда производится изменение фильтра, в том числе тогда, когда его параметры устанавливаются впервые, вызывается метод `Filter_SelectionChanged`. Данный метод принимает текущий фильтр и использует его для выполнения запроса и создания нового списка результатов. Эту логику иллюстрирует листинг 8.2.

Листинг 8.2. Фильтрация отдельных позиций на основе текста запроса

```
private void Filter_SelectionChanged(object sender,
  SelectionChangedEventArgs e)
{
  var query = _query.ToLower();
  var selectedFilter = e.AddedItems.FirstOrDefault() as Filter;
  if (selectedFilter != null)
  {
    selectedFilter.Active = true;
    if (selectedFilter.Name.Equals("All"))
```

```
    {
      DefaultViewModel["Results"] =
        (from g in App.Instance.DataSource.GroupList
          from i in g.Items
          where i.Title.ToLower().Contains(query)
            || i.Description.ToLower().Contains(query)
        select
          new SearchResult
          {
            Image = i.DefaultImageUri,
            Title = i.Title,
            Id = i.Id,
            Description = i.Description
          }).ToList();
      }
      else
      {
        var blogGroup =
          ↳ App.Instance.DataSource.GetGroup(selectedFilter.Id);
        DefaultViewModel["Results"] =
          (from i in blogGroup.Items
          where i.Title.ToLower().Contains(query)
            || i.Description.ToLower().Contains(query)
        select
          new SearchResult
          {
            Id = i.Id,
            Image = i.DefaultImageUri,
            Title = i.Title,
            Description = i.Description
          }).ToList();
      }
      object results;
      ICollection resultsCollection;
      if (DefaultViewModel.TryGetValue("Results", out results) &&
        (resultsCollection = results as ICollection) != null &&
        resultsCollection.Count != 0)
      {
        VisualStateManager.GoToState(this, "ResultsFound", true);
        return;
      }
    }
    VisualStateManager.GoToState(this, "NoResultsFound", true);
}
```

Обратите внимание на то, что мы используем диспетчер визуальных состояний (Visual State Manager, VSM), чтобы изменять состояние в том случае, если соответствий не найдено. Результат поиска по ключевому слову, которое ввел пользователь, показан на рис. 8.5.

Рис. 8.5. Результат поиска приложений

Выполнять поиск приложений вы можете в любое время, вызывая соответствующий интерфейс с помощью клавиатурного сокращения Win+Q. Если приложение уже загружено, вы можете заметить поисковые подсказки, которые появляются по мере ввода текста запроса.

Реализовать эту возможность довольно просто. Нужно лишь подписаться на событие, которое возникает, когда пользователь вводит поисковый запрос. Для данного приложения результаты основаны на поиске частичных совпадений в заголовках постов. Чтобы это сделать, сначала нужно зарегистрироваться для обработки соответствующего события после загрузки контента блога:

```
Windows.ApplicationModel.Search.SearchPane.GetForCurrentView()
  .SuggestionsRequested += SplashPage_SuggestionsRequested;
```

Встроенный механизм поиска запоминает предыдущие поисковые запросы и выводит их в списке предложений. Когда возникает событие, вы можете добавить другие предложения в эту коллекцию. Это могут быть либо предложения по запросам (поисковые запросы, которые могут дать нужные пользователю результаты), либо предложения по результатам (потенциальные соответствия поисковому запросу). Пользовательский интерфейс поиска может выводить на экран до пяти позиций каждого типа. Реализация этого механизма представлена в листинге 8.3. Текст запроса игнорируется, если его длина составляет меньше трех символов. LINQ-запрос фильтрует текст заголовков, отбирая строки, которые начинаются с введенных пользователем символов, а затем приводит в порядок результаты, убирая пустые строки и специальные символы, чтобы получился набор уникальных слов, содержащихся в заголовках.

Листинг 8.3. Реализация поисковых предложений

```
void SplashPage_SuggestionsRequested(Windows.ApplicationModel.
Search
    .SearchPane sender,
   Windows.ApplicationModel.Search
    .SearchPaneSuggestionsRequestedEventArgs args)
{
  var query = args.QueryText.ToLower();
  if (query.Length < 3) return;
    var suggestions = (from g in App.Instance.DataSource.GroupList
                       from i in g.Items
                       from keywords in i.Title.Split(' ')
                       let keyword = Regex.Replace(
                         keywords.ToLower(), @"[^\w\.@-]", "")
                       where i.Title.ToLower().Contains(query)
                       && keyword.StartsWith(query)
                       orderby keyword
                       select keyword).Distinct();

    args.Request.SearchSuggestionCollection
      .AppendQuerySuggestions(suggestions);
}
```

Последнее, что нам осталось сделать, — это обработать выбор пользователем искомых позиций. Вы можете заметить, что XAML-разметка страницы поиска — это сетка, которая поддерживает выделение позиции с помощью следующего атрибута:

```
IsItemClickEnabled="True"
```

Данный атрибут позволяет выделять позиции щелчком мыши или прикосновением. Для того чтобы обработать событие выделения, добавим обработчик события `ItemClick`:

```
ItemClick="ResultsGridView_ItemClick_1"
```

При обработке события производится переход на страницу подробной информации по выбранной позиции:

```
private void ResultsGridView_ItemClick_1(object sender,
  ItemClickEventArgs e)
{
  Navigate(typeof (ItemDetailPage),
    ((SearchResult) e.ClickedItem).Id);
}
```

Подводя итоги, хочу отметить, что обслуживание контракта поиска выполняется поэтапно. Хотя вы должны обработать поисковый запрос и определить, следует ли выполнять непосредственный переход к странице поиска или использовать для предварительной загрузки данных возможности расширенной экранной заставки, шаблон предоставляет все внутренние механизмы для реализации поиска. После того как получены результаты поиска, вы создаете поисковые фильтры и выводите результаты поиска, в полном объеме удовлетворяющие всем установленным фильтрам. И наконец, обработка выбора пользователем найденной позиции подразумевает организацию перехода на соответствующую страницу выбранного приложения.

Общий доступ

Сейчас, когда в Интернете чрезвычайно популярны социальные сети, обмен данными актуален как никогда. В предыдущих версиях Windows основным инструментом передачи данных между приложениями был буфер обмена. Если вам нужно было создать приложение, способное отправлять что-либо в Интернет, это означало необходимость самостоятельного создания всего необходимого, в том числе пользовательского интерфейса и программной логики взаимодействия с прикладными программными интерфейсами различных социальных сетей. Windows 8 радикально меняет ситуацию, предоставляя стандартный прикладной программный интерфейс для организации обмена данными различных типов: от обычных текстов до изображений и даже до объектов файловой системы.

Контракт Share (Общий доступ) позволяет приложениям обмениваться данными стандартным способом, не задумываясь о том, какие именно приложения установлены в системе и как они будут обрабатывать общие данные. Ваше приложение может работать в качестве приложения-источника, предоставляя контент, который пользователь может захотеть передать в общий доступ. Приложение может также выступать в роли приложения-приемника, то есть получать контент от других приложений. Взаимодействие в рамках контракта общего доступа производится посредством WinRT-компонента `DataTransferManager` из пространства имен `Windows.ApplicationModel.DataTransfer`. Всю документацию по этому пространству имен можно найти на странице http://msdn.microsoft.com/ru-ru/library/windows/apps/windows.applicationmodel.datatransfer.aspx.

Источник контента для общего доступа

Приложение Wintellog3 выступает в качестве источника и может передавать в общий доступ подробные данные о блогах, постах и изображениях. Чтобы успешно реализовать общий доступ, в системе должны быть установлены приложения, способные принимать контент. Приложение ImageHelper2 из главы 3 может принимать изображения. Почтовый клиент способен принимать гиперссылки и тексты. Советую вам также загрузить, скомпилировать и установить приложение Sharing content target app sample из примеров SDK Windows 8, которые можно найти на странице http://code.msdn.microsoft.com/windowsapps/Sharing-Content-Target-App-e2689782/.

Это приложение спроектировано так, чтобы принимать предоставляемый в общий доступ контент любого типа. Когда пользователь нажимает на чудо-кнопку Share (Общий доступ), вызывается событие `DataRequested` компонента `DataTransferManager`. Класс `App` содержит метод для подписки на это событие:

```
public void RegisterForShare()
{
  var dataManager = DataTransferManager.GetForCurrentView();
  dataManager.DataRequested += DataManager_DataRequested;
}
```

Данное событие возникает со ссылкой на компонент `DataTransferManager` и с параметром типа `DataRequestedEventArgs`. Параметр содержит свойство `Request` типа `DataRequest`, которое, в свою очередь, предоставляет доступ к объекту `DataPackage` (пакет данных). Этот объект может служить контейнером для любых данных, которые вы хотите передать другим приложениям. В табл. 8.3 перечислены типы данных, которые можно хранить в пакете данных.

Таблица 8.3. Типы данных, которые поддерживает класс DataPackage

Тип	Описание
Bitmap	Битовый образ
Data	Нестандартный формат данных, сохраненных как формат JSON на основе схемы, определенной на странице http://schema.org/, или на основе нестандартной схемы, которая должна быть опубликована и доступна для реализации другими приложениями
HTML	Контент на языке разметки гипертекста
RTF	Форматированный текст
Storage items	Список объектов типа IStorageItem, используемый для передачи файлов между приложениями
Text	Обычный текст
URI	Универсальный идентификатор ресурса

Кроме того, пакет данных поддерживает асинхронные запросы при условии установки соответствующего поставщика данных. Поставщик сообщает о типе предоставляемой информации и выделяет функцию обратного вызова, которая вызывается, когда целевое приложение будет готово к приему данных. Подобный подход используется тогда, когда требуется много времени на пакетирование и доставку.

Вместе с реальным контентом можно передать графическую миниатюру, представляющую данные, заголовок и описание. Для реализации общего доступа главный класс приложения `App` предоставляет свойство, позволяющее каждой отдельной странице получать обработчик общего доступа и, следовательно, нужный контекст:

```
public Action<DataTransferManager, DataRequestedEventArgs>
  Share { get; set; }
```

Страницы, которые не поддерживают общий доступ, когда пользователь переходит к ним, устанавливают это свойство в `null`. Например, так происходит при переходе на страницу `GroupedItemsPage`:

```
protected override void LoadState(Object navigationParameter,
  Dictionary<String, Object> pageState)
{
  App.Instance.Share = null;
  DefaultViewModel["Groups"] = App.Instance.DataSource.GroupList;
  groupGridView.ItemsSource =
    groupedItemsViewSource.View.CollectionGroups;
}
```

Обработчик самостоятельно проверяет, был ли зарегистрирован обратный вызов:

```
void DataManager_DataRequested(DataTransferManager sender,
  DataRequestedEventArgs args)
{
  if (Share != null)
  {
    Share(sender, args);
  }
}
```

Если данные в общий доступ не предоставляются, пользователь информируется об этом. Вы можете задать текст этого сообщения, указав причину, по которой операция общего доступа не удалась. Следующий код добавляется после проверки на наличие обратного вызова:

```
else
{
  args.Request
    .FailWithDisplayText(
       "Please choose a blog or item to enable sharing.");
}
```

Результат такой попытки общего доступа показан на рис. 8.6. В данном сообщении пользователю предлагается выбрать другой блог или пост, чтобы операция общего доступа стала возможной.

Страницы, поддерживающие общий доступ, регистрируют метод для обработки обратного вызова:

```
App.Instance.Share = Share;
```

Если пользователь переходит к странице блога, в общий доступ будет предоставлен URL-адрес блога. Сначала производится проверка, позволяющая выяснить, действительно ли текущий блог существует:

```
var group = DefaultViewModel["Group"] as BlogGroup;
if (group == null)
{
  return;
}
```

Далее вместе с URL-адресом блога к передаче подготавливаются заголовок и описание.

```
 dataRequestedEventArgs.Request.Data.Properties.Title = group.Title;
dataRequestedEventArgs.Request.Data.SetUri(group.RssUri);
dataRequestedEventArgs.Request.Data.Properties.Description =
  "Wintellog RSS feed.";
```

Страница `ItemDetailPage` содержит более сложную логику общего доступа. Если пользователь просто нажмет чудо-кнопку Share (Общий доступ), не выделив никакого текста, приложение передаст в общий доступ ссылку на пост блога вместе со всем его контентом:

```
dataRequestedEventArgs.Request.Data.SetText(item.Description);
```

Если в данном посте блога имеется изображение, оно пакетируется в качестве битового образа:

```
if (item.DefaultImageUri != null)
{
  dataRequestedEventArgs.Request.Data.SetBitmap(
    RandomAccessStreamReference
       .CreateFromUri(item.DefaultImageUri));
}
```

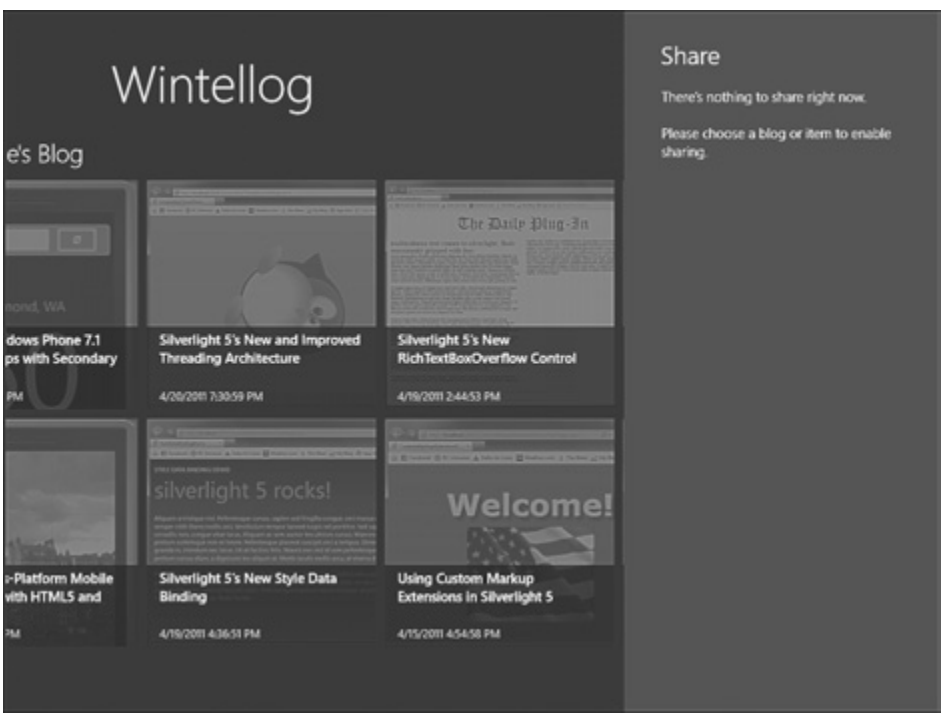


Рис. 8.6. Попытка общего доступа не удалась, о чем сообщено пользователю

Данные поста блога также пакетируются с применением нестандартного формата данных.

Нестандартные данные

Нестандартные данные приходится пакетировать по одной из хорошо известных схем, которые предлагаются на странице http://schema.org/, либо использовать собственную схему. Схема пакетирования постов блогов есть на странице http://schema.org/BlogPosting. То есть любое приложение, которое распознает данную схему, может обработать информацию, выделенную в общий доступ с ее помощью.

Нестандартные данные обычно хранят в формате JSON. Из-за ограничений встроенного класса `DataContractJsonSerializer` (он не предоставляет стандартизированного формата и требует наличия атрибутов для успешной сериализации экземпляра) я решил использовать популярную открытую JSON.NET-библиотеку, предложенную Джеймсом Ньютоном-Кингом (James Newton-King). Подробности о ней можно узнать на странице http://james.newtonking.com/projects/json-net.aspx.

Библиотека пакетируется с использованием расширения NuGet (http://nuget.org), поэтому установить ее не сложнее, чем открыть диспетчер пакетов с помощью команды Tools ▸ Library Package Manager ▸ Package Manager Console (Сервис ▸ Диспетчер пакетов библиотек ▸ Консоль диспетчера пакетов) и ввести следующий текст (обратите внимание, что имя пакета чувствительно к регистру):

```
install-package Newtonsoft.Json
```

Эта команда приведет к автоматической загрузке всех необходимых файлов и добавлению ссылки в текущий проект. В учебном приложении это уже сделано. С помощью библиотеки Json.NET я могу создать анонимный тип для хранения информации блога и сериализовать его в формат JSON с помощью одной строки кода, как показано в листинге 8.4.

Листинг 8.4. Сериализация динамического типа данных с помощью Json.NET

```
private static async Task<string> CustomData(BlogItem item)
{
  var schema = new
  {
    type = "http://shema.org/BlogPosting",
    properties = new
    {
      description = string.Format(
```

продолжение ⇩

Листинг 8.4 *(продолжение)*

```
      "Blog post from {0}.",
      item.Group.Title),
      image = item.DefaultImageUri,
      name = item.Title,
      url = item.PageUri,
      audience = "Windows 8 Developers",
      datePublished = item.PostDate,
      headline = item.Title,
      articleBody = item.Description
    }
  };
  return await JsonConvert.SerializeObjectAsync(schema);
}
```

Для того чтобы пакетировать нестандартные данные, вызывается метод `SetData`, которому передается схема и JSON-контент:

```
var data = await CustomData(item);
dataRequestedEventArgs.Request.Data.SetData(
  "http://schema.org/BlogPosting", data);
```

Существуют сотни схем, которые можно использовать для реализации общего доступа приложений к форматированным данным и другому контенту. Если вы хотите предложить собственную схему, вам достаточно предоставить уникальный URL-адрес, описывающий формат схемы, и затем упаковать данные с ее помощью. Любое приложение, осведомленное о данной схеме, сможет обработать данные, предварительно проверив тип схемы. О том, как принимать и обрабатывать данные, предоставляемые в общий доступ, вы узнаете далее в этой главе.

Выделение текста

Что, если пользователь хочет предоставить в общий доступ лишь небольшую выдержку из поста блога? Выделение текста построено на базе существующих текстовых XAML-элементов управления. Контент поста блога находится в элементе управления `RichTextBlock`. Этот элемент управления предоставляет событие `SelectionChanged`:

```
<RichTextBlock x:Name="richTextBlock" Width="560"
  Style="{StaticResource ItemRichTextStyle}"
  SelectionChanged="RichTextBlock_SelectionChanged_1">
```

Когда возникает это событие, выделенный текст сохраняется в локальной переменной:

```
private void RichTextBlock_SelectionChanged_1(object sender,
  RoutedEventArgs e)
{
  var richTextControl = sender as RichTextBlock;
  _selection = richTextControl != null ?
    richTextControl.SelectedText : string.Empty;
}
```

Если после выделения текста на странице активировать чудо-кнопку Share (Общий доступ), этот текст будет обработан следующим образом:

```
dataRequestedEventArgs.Request.Data.Properties.Title =
  string.Format("Excerpt from {0}", item.Title);
dataRequestedEventArgs.Request.Data.Properties.Description =
  string.Format("An excerpt from the {0} blog at {1}.",
  item.Group.Title, item.PageUri);
dataRequestedEventArgs.Request.Data.SetText(
  string.Format("{0}\r\n\r\n{1}", _selection, item.Group.RssUri));
```

На рис. 8.7 показано, что произойдет после выделения в посте блога отдельного предложения и предоставление его в общий доступ учебному приложению, о чем мы говорили ранее.

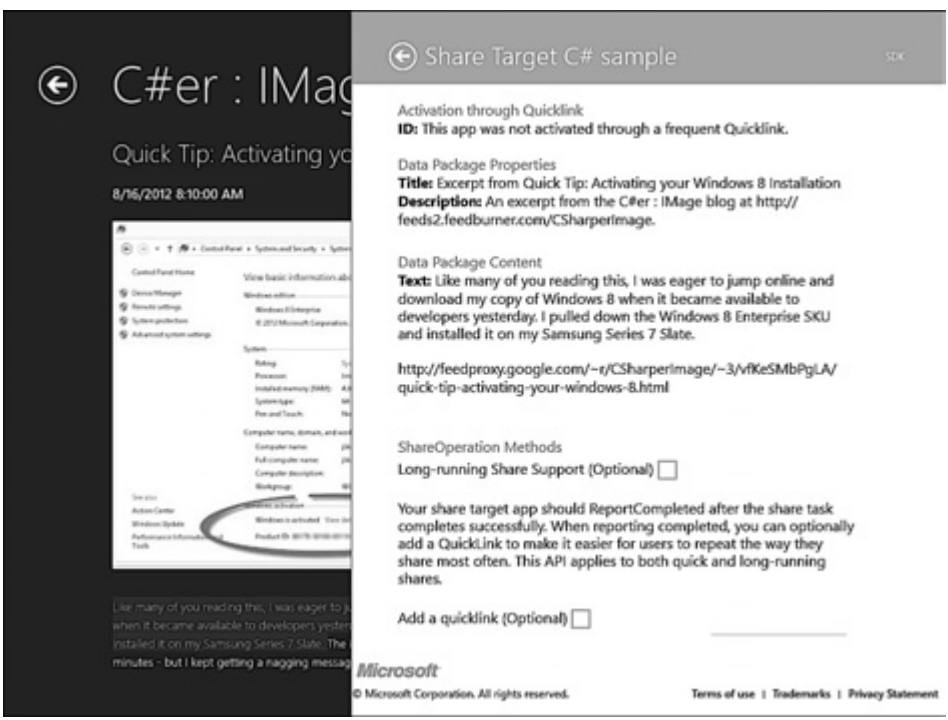


Рис. 8.7. Предоставление в общий доступ части поста

Включение в текст вместо самого пакета URL-адреса поста — это пример данных, предоставляемых в общий доступ с особой целью. Когда пользователь выделяет фрагмент текста, он собирается предоставить в общий доступ именно этот фрагмент, а не всю страницу. Предоставление в общий доступ URL-адреса источника в главном пакете может привести к тому, что другие приложения загрузят страницу целиком, хотя изначальной целью было предоставление небольшого фрагмента текста. То есть только предоставляемый в общий доступ контент должен составлять текстовый фрагмент пакета данных.

Получение контента приложением-приемником

Получение контента находится на противоположной стороне механизма общего доступа. Упомянутое ранее учебное приложение демонстрирует несколько вариантов получения контента, выделенного в общий доступ. Для того чтобы приложение смогло выступать в роли приложения-приемника, нужно щелкнуть правой кнопкой мыши на проекте в обозревателе решений (Solution Explorer) и выбрать в появившемся меню команду Add ▸ New Item (Добавить ▸ Создать элемент). После этого нужно выбрать шаблон Share Target Contract (Совместный доступ к целевому контракту), как показано на рис. 8.8.

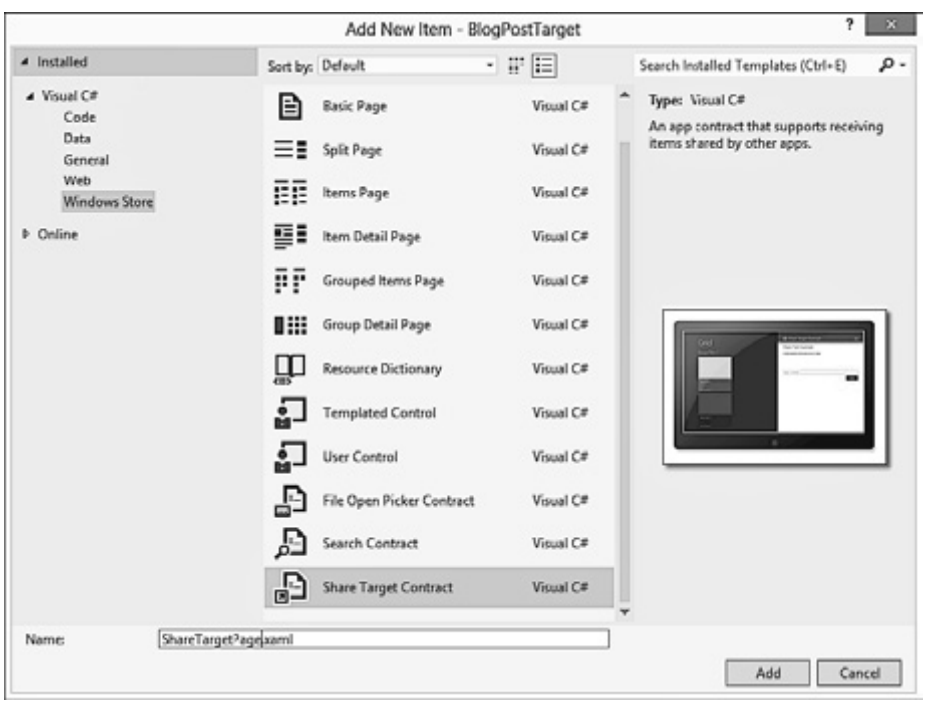


Рис. 8.8. Добавление в приложение поддержки контракта общего доступа

Шаблон призван решить несколько задач. Так, будет добавлена страница, которая сыграет роль целевого приложения для операций общего доступа. На этой странице-приглашении пользователю будет предлагаться ввести сведения, необходимые для успешного завершения операции общего доступа. Например, если у вас установлено приложение Puzzle Touch (это бесплатное приложение можно загрузить из Магазина Windows) и вы собираетесь передать ему изображение, страница-приглашение предложит указать размер и сложность головоломки (рис. 8.9).

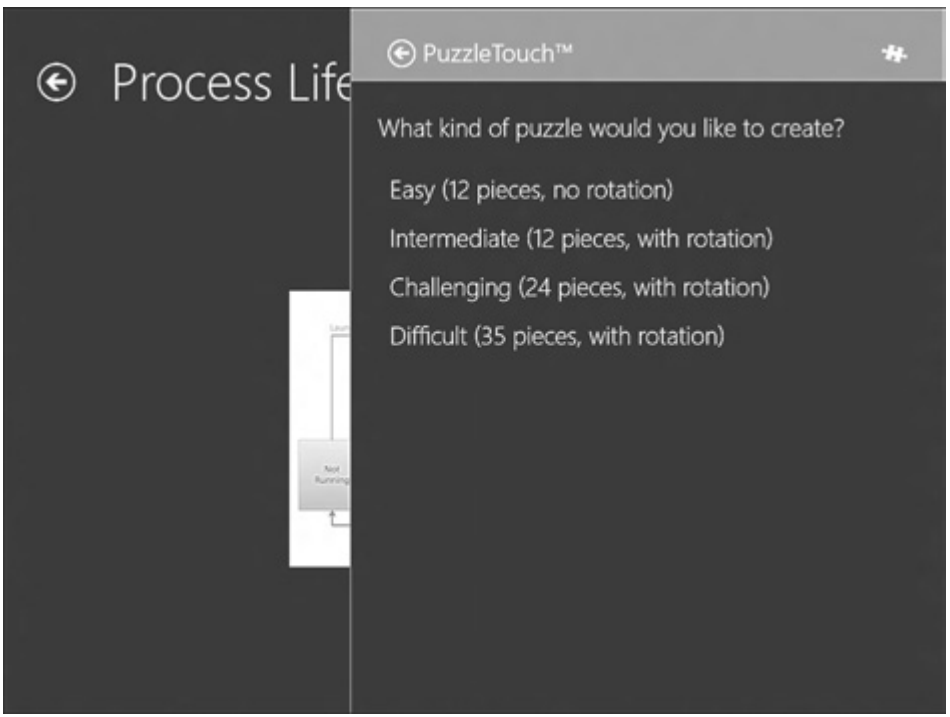


Рис. 8.9. Страница-приглашение в приложении Puzzle Touch

Шаблон также обновит манифест приложения. В него будет добавлено объявление целевого приложения, причем по умолчанию поддерживаются форматы данных `text` и `uri`. В итоге на экран будут выведены заголовок и описание, приглашающие пользователя ввести комментарии, и кнопка, призванная завершить операцию общего доступа. Результат показан на рис. 8.10.

Приложение BlogPostTarget из учебных приложений для главы 8 показывает, как обрабатывать нестандартные данные. Когда начинается операция общего доступа, приложение активизируется благодаря контракту общего доступа. Код для обработки данного варианта активизации приложения автоматически добавляется в класс `App` благодаря шаблону:

```
protected override void OnShareTargetActivated(
  ShareTargetActivatedEventArgs args)
{
  var shareTargetPage = new ShareTargetPage();
  shareTargetPage.Activate(args);
}
```

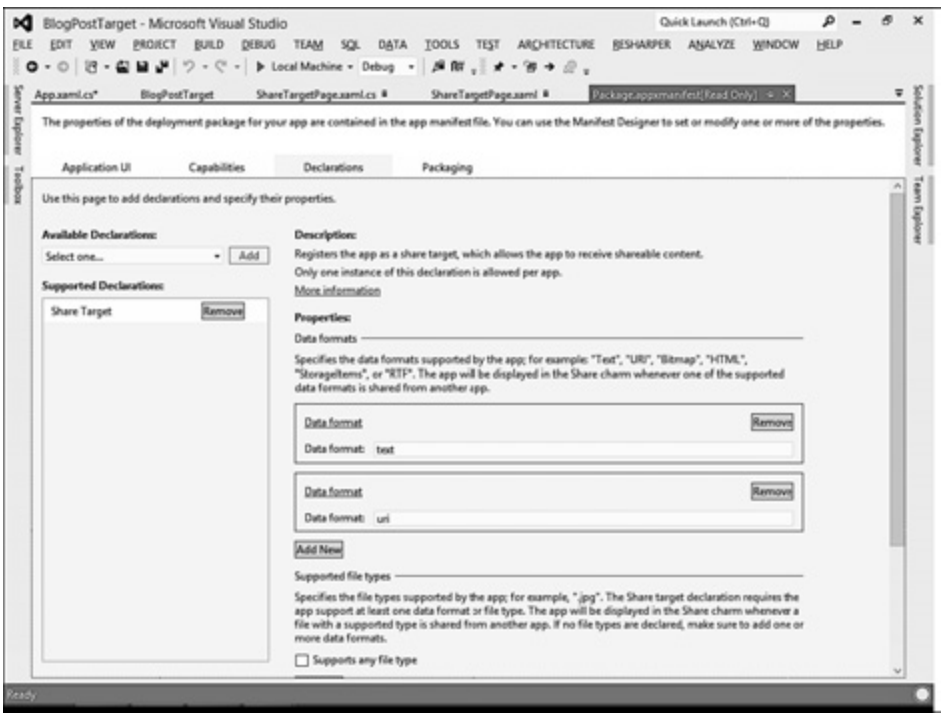


Рис. 8.10. Настройка манифеста для целевого приложения

Метод `Activate` элемента управления `ShareTargetPane` выполняет загрузку информации из пакета данных. В листинге 8.5 показан код, предлагаемый по умолчанию для получения и установки различных свойств с целью привязки данных.

Листинг 8.5. Шаблонный код для загрузки контента в операции общего доступа

```
_shareOperation = args.ShareOperation;
DataPackagePropertySetView shareProperties =
  _shareOperation.Data.Properties;
var thumbnailImage = new BitmapImage();
DefaultViewModel["Title"] = shareProperties.Title;
```

```
DefaultViewModel["Description"] = shareProperties.Description;
DefaultViewModel["Image"] = thumbnailImage;
DefaultViewModel["Sharing"] = false;
DefaultViewModel["ShowImage"] = false;
DefaultViewModel["Comment"] = String.Empty;
DefaultViewModel["SupportsComment"] = true;
Window.Current.Content = this;
Window.Current.Activate();
if (shareProperties.Thumbnail != null)
{
  var stream = await shareProperties.Thumbnail.OpenReadAsync();
  thumbnailImage.SetSource(stream);
  DefaultViewModel["ShowImage"] = true;
}
```

Для того чтобы принять данные нестандартного формата, которые передает приложение Wintellog3, в коде приложения BlogPostTarget производится проверка, присутствуют ли данные этого формата в пакете. При этом используется тот же идентификатор схемы, который применялся при записи данных в пакет:

```
const string BLOG_POST = "http://schema.org/BlogPosting";
if (!_shareOperation.Data.Contains(BLOG_POST)) return;
```

Если необходимые данные в пакете есть, производится их извлечение и проверка, не содержит ли пакет пустой набор данных:

```
var data = await _shareOperation.Data.GetDataAsync(BLOG_POST);
if (data == null) return
```

И наконец, данные десериализуются в исходный формат и сохраняются. Тот же самый инструмент, который использовался для сериализации данных (Json.NET), требуется и для обратной операции. В примере задействован анонимный тип, поэтому шаблон передается библиотеке Json.NET. Шаблон отражает те данные, которые были записаны в пакет приложением Wintellog3. Описанные действия иллюстрирует листинг 8.6.

Листинг 8.6. Загрузка данных нестандартного формата в операции общего доступа

```
DefaultViewModel["ShowBlog"] = true;
DefaultViewModel["BlogPost"] = Newtonsoft.Json.JsonConvert
  .DeserializeAnonymousType((string) data,
  new
  {
```

продолжение ⇩

Листинг 8.6 *(продолжение)*

```
    type = "http://shema.org/BlogPosting",
    properties = new
    {
      description = string.Empty,
      image = new Uri("http://schema.org/"),
    name = string.Empty,
    url = new Uri("http://schema.org/"),
    audience = "Windows 8 Developers",
    datePublished = DateTime.Now,
    headline = string.Empty,
    articleBody = string.Empty
  }
});
```

Когда нестандартная информация поста блога извлечена, чтобы показать ее конечному пользователю, применяется механизм привязки данных. В листинге 8.7 показан XAML-код, служащий для вывода заголовка и прокручиваемой области с материалом статьи. На заголовке можно щелкнуть, что приводит к открытию поста в браузере, предлагаемом по умолчанию (обычно это Internet Explorer).

Листинг 8.7. XAML-разметка для вывода поста блога

```
<StackPanel Visibility="{Binding ShowBlog,
       ↳ Converter={StaticResource BooleanToVisibilityConverter}}"
      Orientation="Vertical">
  <HyperlinkButton NavigateUri="{Binding BlogPost.properties.url}"
    Content="{Binding BlogPost.properties.headline}"/>
  <ScrollViewer Height="200">
    <RichTextBlock Margin="20">
      <Paragraph>
        <Run FontWeight="SemiLight"
          FontSize="13"
          Text="{Binding BlogPost.properties.articleBody}"/>
      </Paragraph>
    </RichTextBlock>
  </ScrollViewer>
</StackPanel>
```

На рис. 8.11 вы можете видеть результат операции общего доступа. Обратите внимание, что на экран выводятся заголовок и описание, за которыми следуют гиперссылка и контент статьи. Программа сможет успешно выводить это контент, получаемый от любого другого приложения, использующего ту же схему.

Рис. 8.11. Предоставление поста блога в общий доступ

Контракт общего доступа — это мощное средство, которое позволяет организовать обмен контентом между приложениями. Ваше приложение может предоставить информацию в общий доступ, не имея сведений о том, какие еще программы установлены в системе. Это позволяет поддерживать расширенные сценарии работы, такие как отправка данных в Twitter или Facebook, не задумываясь о самостоятельной реализации механизмов взаимодействия с этими службами. Когда пользователь в качестве приложения-приемника устанавливает у себя программу, поддерживающую ту или иную социальную сеть, ваше приложение находится буквально в нескольких шагах от возможности отправки контента в эту социальную сеть.

Параметры

Контракт Settings (Параметры) позволяет единообразно представлять параметры приложения и сведения о нем. О том, как настраивать приложение, вы узнали из главы 4. В рассматриваемом там примере применялся нестандартный элемент управления, поддерживающий анимацию панели. Здесь мы узнаем, как применять набор инструментов с открытым исходным

кодом, который позволяет значительно упростить создание подобной панели. Я имею в виду набор Callisto, созданный сотрудником Microsoft Тимом Хейером (Tim Heuer). Найти его можно на странице http://timheuer.com/blog/archive/2012/05/31/introducing-callisto-a-xaml-toolkit-for-metro-apps.aspx.

Для того чтобы начать использовать пакет Callisto, его нужно установить с помощью расширения NuGet. Как работать с консолью диспетчера пакетов, вы уже знаете. Для этого есть также графический интерфейс. Для вызова этого интерфейса нужно выбрать команду Tools ▸ Manage NuGet Packages for Solution (Сервис ▸ Управление пакетами NuGet для решения). В левой части появившегося окна выберите пункт Online (В сети) и в поле поиска введите название `Callisto`. Когда пакет будет найден, щелкните на кнопке Install (Установить), как показано на рис. 8.12. В результате пакет загрузится и ссылка на него будет добавлена в проект. Этот пакет вы также можете загрузить самостоятельно со страницы https://github.com/timheuer/callisto.

Рис. 8.12. Использование средства управления пакетами NuGet для установки Callisto

После того как пакет Callisto установлен, вы можете быстро и просто добавить в приложение панель параметров. Так, в нашем случае сначала был добавлен элемент управления с именем WintellogSettings. Его XAML-разметка содержит простой список, который выводит на экран имя приложения, гиперссылку для перехода на сайт с его исходным кодом и кнопку для получения URI-идентификатора канала. Теперь все это доступно из панели

параметров, а не из панели приложения, как это было ранее. В листинге 8.8 показана XAML-разметка панели параметров.

Листинг 8.8. Простая XAML-разметка панели параметров

```
<Grid Style="{StaticResource LayoutRootStyle}">
  <StackPanel Orientation="Vertical">
    <TextBlock Style="{StaticResource BodyTextStyle}"
      Text="Wintellog by Jeremy Likness"
      TextWrapping="Wrap"/>
    <HyperlinkButton
      NavigateUri="http://windows8applications.codeplex.com/"
      Content="Click to Access Source Code"/>
    <Button Content="Tap to Copy Channel URI"
      Click="Button_Click_1"/>
  </StackPanel>
</Grid>
```

Вспомогательный код (code-behind) будет обрабатывать нажатие кнопки для получения URI-идентификатора канала и его вывода на экран (листинг 8.9).

Листинг 8.9. Код для вывода URI-идентификатора канала

```
private async void Button_Click_1(object sender, RoutedEventArgs e)
{
  var message = App.Instance.Channel != null
    ? "The channel has been copied to the clipboard."
    : string.Format("Error: {0}", App.Instance.ChannelError);
  var dataPackage = new DataPackage();
  if (App.Instance.Channel != null)
  {
    dataPackage.SetText(App.Instance.Channel.Uri);
  }
  Clipboard.SetContent(dataPackage);
  var dialog = new MessageDialog(message);
  await dialog.ShowAsync();
}
```

В классе `App` при запуске приложения регистрируется обработчик события `CommandRequested`:

```
private void RegisterSettings()
{
  var pane = SettingsPane.GetForCurrentView();
  pane.CommandsRequested += Pane_CommandsRequested;
}
```

Этот обработчик события регистрирует пункт меню About (О программе):

```
void Pane_CommandsRequested(SettingsPane sender,
  SettingsPaneCommandsRequestedEventArgs args)
{
  var aboutCommand = new SettingsCommand("About", "About",
    SettingsHandler);
  args.Request.ApplicationCommands.Add(aboutCommand);
}
```

И наконец, обработчик использует класс `SettingsFlyout` из пакета Callisto для вывода панели на экран. Соответствующий код приведен в листинге 8.10.

Листинг 8.10. Использование класса SettingsFlyout из пакета Callisto

```
private SettingsFlyout _flyout;
private void SettingsHandler(IUICommand command)
{
  _flyout = new SettingsFlyout
  {
    HeaderText = "About",
    Content = new WintellogSettings(),
    IsOpen = true
  };
  _flyout.Closed += (o, e) => _flyout = null;
}
```

Обратите внимание на то, что применение специального элемента управления значительно упрощает операцию. Она сводится к указанию заголовка, назначению контента (в данном случае это пользовательский элемент управления) и включению режима вывода элемента. Код также очищает любые ссылки на элемент управления, когда панель параметров закрывается. Результат открытия новой страницы с информацией о программе с помощью чудо-кнопки Settings (Параметры) представлен на рис. 8.13.

Этот простой пример показывает, как работать с гиперссылками и кнопками, но тот же подход пригоден и для организации сложной системы настройки приложения. Достаточно создать пользовательский интерфейс для настройки и затем сохранить заданные параметры с помощью методов, о которых вы узнали в главе 6.

Рис. 8.13. Страница сведений о программе

Выводы

В этой главе вы узнали о том, как усовершенствовать ваше приложение с помощью контрактов, которые являются частью Windows 8. Контракты предоставляют приложениям цельный стандартизированный механизм взаимодействия друг с другом и с платформой Windows 8. При создании приложений вы можете задействовать встроенные возможности поиска и передачи между приложениями форматированного текста, изображений и других данных. Также вы можете предоставить конечному пользователю привычную для него панель параметров, позволяющую настраивать приложение.

В следующей главе вы узнаете о том, как лучше организовать код с помощью паттерна MVVM. Среди многих достоинств данного паттерна можно отметить улучшенные возможности тестирования. Я объясню, почему так важно тестирование, и покажу, как разрабатывать автоматические модульные тесты для проверки приложений. Вы узнаете, как паттерн MVVM способен помочь в разработке приложений с помощью других технологий, таких как WPF и Silverlight, причем код этих приложений может использоваться в программных продуктах для Windows 8.

MVVM и тестирование

9

MVVM — это аббревиатура от Model-View-View Model. Так называется паттерн дизайна пользовательского интерфейса. Это тот самый паттерн (решение, ориентированное на многократное использование), который был создан много лет назад для разработки WPF-приложений. Позже с помощью технологии Silverlight он был доработан, чтобы поддерживать веб-приложения для платформ knockout.js, Windows Phone, Xbox и Windows 8. Основная идея, лежащая в основе MVVM, заключается в том, чтобы отделить код реализации пользовательского интерфейса от кода логики уровня представления и кода бизнес-логики приложения, при этом взаимодействие уровней в модели реализуется через привязки данных. Подобная модель построения приложений получает массу преимуществ. В частности, упрощается работа над дизайном приложений, то есть над тем, как они выглядят и как воспринимаются пользователем, упрощается тестирование. С паттерном MVVM для написания компонентов и тестов можно использовать переносимую библиотеку классов (Portable Class Library, PCL). Готовые компоненты и тесты могут совместно работать в средах WPF, Silverlight и в приложениях для Windows 8. О PCL рассказывается в этой главе далее.

Возможно, вы уже знаете, что в Visual Studio 2012 XAML-шаблоны на C# уже имеют встроенную поддержку MVVM. Класс `LayoutAwarePage` содержит словарное свойство `DefaultViewModel`. Данные (включая классы), которые реализуют уведомления об изменении свойств, привязаны к словарю, и на них можно ссылаться из кода. Применение MVVM в приложениях для Windows 8 весьма желательно, так как позволяет задействовать

возможности XAML в плане привязки данных, о которых вы узнали в главе 3. Еще одно преимущество паттерна MVVM заключается в том, что он помогает упростить тестирование.

Тестирование — это то, что многие разработчики любят или ненавидят. В этой главе я надеюсь показать, что верно проведенное модульное тестирование способно сберечь много времени и улучшить возможности приложения в плане его поддержки и расширения. MVVM предоставляет хорошую основу для тестирования, так как четко выделенный код — это еще и код, для которого легко писать тесты.

Правильное использование MVVM вкупе с тестированием упрощает создание приложений, особенно если у вас большая команда разработчиков или разные команды дизайнеров и программистов. Применение модульных тестов также помогает сократить количество инцидентов с потребителями, так как ошибки выявляются гораздо раньше, еще на этапе разработки. Модульное тестирование упрощает расширение и переделку приложений, да и сам по себе паттерн MVVM позволяет делать то, что я называю *изоляцией переделок* (refactoring isolation). То есть благодаря ему можно модифицировать части приложения без необходимости в качестве побочного эффекта изменений обновлять каждый модифицированный модуль.

Паттерны дизайна пользовательского интерфейса

Паттерны UI-дизайна — это ориентированные на многократное использование решения, которые призваны решить проблему поддержки уровня представления приложений независимо от логики нижележащих уровней, бизнес-логики, служб и данных. Вот некоторые из проблем, которые могут быть решены:

- **Гибкость пользовательского интерфейса.** Часто со временем пользовательский интерфейс приложений может существенно меняться. Меняется его внешний вид, ощущения пользователя, способы взаимодействия. Хорошо спроектированный и качественно реализованный паттерн UI-дизайна может свести к минимуму воздействие подобных изменений на бизнес-логику и данные приложения. Этот подход прослеживается в существующих шаблонах, которые предлагают различные варианты представления данных (режим фиксации, альбомная или портретная ориентация) без необходимости вносить изменения в нижележащие классы, предоставляющие эти данные.

- **Параллельные процессы программирования и разработки дизайна.** Часто команда дизайнеров, которая может состоять из нескольких специалистов, отделена от команды программистов. При этом обе команды обладают совершенно разными знаниями и навыками. Паттерны UI-дизайна повышают эффективность совместной работы таких команд, позволяя четко разделять их рабочие процессы. В итоге программисты и дизайнеры получают возможность работать параллельно с минимумом конфликтных ситуаций. Данные, предназначенные для дизайнеров, позволяют им работать непосредственно с XAML-разметкой, реализуя желаемые визуальные и тактильные качества приложения. При этом они не знают или непосредственно не касаются нижележащей логики приложения.
- **Выделение логики уровня представления.** На уровне представления существуют общие паттерны, например паттерн списка позиций, в котором пользователь может выбрать только одну позицию, позволяет представлять разные UI-элементы (комбинированный список, сетка, обычный список и т. д.). Паттерны UI-дизайна дают возможность отделить элементы данных от реализации уровня представления, в результате базовая функциональность остается прежней независимо от того, как представляется паттерн.
- **Тестирование логики представления.** Комплекс логики представления данных часто относится к ведению программистов. Тестирование алгоритмов упрощается благодаря тому, что для этого нет необходимости в готовом пользовательском интерфейсе. В качестве примера можно привести зависимые, или каскадные, списки. Выбор той или иной позиции в первом списке определяет содержимое второго. В идеале у вас должна быть возможность реализовать и протестировать данный режим работы, не рисуя на экране соответствующий элемент управления в виде комбинированного списка и не обрабатывая событий выбора позиций.

В 2005 году программист Джон Госсман (John Gossman) из компании WPF, которая тогда носила кодовое название Avalon, опубликовал пост, в котором фактически представил миру паттерн MVVM (http://blogs.msdn.com/b/johngossman/archive/2005/10/08/478683.aspx). В этом посте он описал «разновидность паттерна MVC (Model/View/Controller — Модель/Представление/Поведение), ориентированную на платформы разработки современного пользовательского интерфейса, где за представление отвечает дизайнер, а не классический программист».

Предисловие к его посту позволяет понять изначальный мотив создания этого паттерна: организация рабочего процесса дизайнеров и программистов. Далее в материале поясняется, что представление задается

декларативно (ссылка на XAML) и на него возлагаются ввод данных, клавиатурные сокращения, визуальные элементы и пр. Модель представления ответственна за решение задач, которые слишком специфичны для общей модели обработки (таких, как сложные UI-операции), за поддержание состояния представления и за проецирование модели на представление, особенно в ситуации, когда модель содержит типы данных, непосредственно не отображающиеся на элементы управления.

Используя привязки данных и диспетчер визуальных состояний (Visual State Manager, VSM), паттерн MVVM поддерживает взаимодействие между реализацией пользовательского интерфейса, с одной стороны, и бизнес-логикой и логикой уровня представления — с другой. Представление управляет моделью представления не через события, а через привязки данных, причем независимо от того, что является причиной: обновление значения, которое, в свою очередь, должно быть синхронизировано с неким свойством модели представления, или отображение некоего события на команду, что-то запускающую в модели представления. Логика уровня представления существует в модели представления в виде кода, реализующего режимы работы, триггеры, визуальные состояния и конвертеры значений.

В приложениях на основе встроенных в Visual Studio 2012 шаблонах класс `LayoutAwarePage` автоматически обнаруживает изменения в визуальном состоянии (например, переход в режим фиксации или изменение ориентации с портретной на альбомную), а метод `InvalidateVisualState` использует VSM для установки нового визуального состояния:

```
string visualState = DetermineVisualState(ApplicationView.Value);
foreach (var layoutAwareControl in this._layoutAwareControls)
{
  VisualStateManager.GoToState(layoutAwareControl,
  visualState, false);
}
```

В XAML-разметке VSM позволяет задавать необходимые шаблоны для различных состояний. Вы уже знаете, как обеспечивать поддержку режима фиксации или как переключиться с элемента управления `GridView` на элемент управления `ListView` при изменении ориентации экрана. Дизайн описывается в XAML-разметке независимо от нижележащего кода, что позволяет добиться четкого разделения ролей.

Вы можете увидеть, на что это похоже, взглянув на рис. 9.1. Обратите внимание, что класс `LayoutAwarePage` поддерживает словарь модели представления, обрабатывает изменения с помощью VSM и является базовым классом, от которого наследуется XAML-разметка для представления. Работа класса `BlogDataSource` координируется с приложением для получения

данных и последующего их вывода посредством словаря модели представления, начиная со свойства `GroupList`.

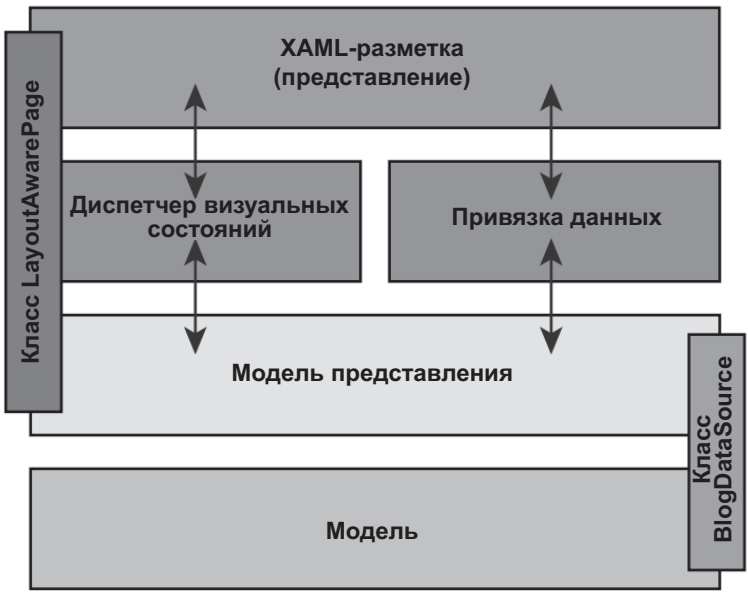


Рис. 9.1. Паттерн MVVM в приложении Wintellog

Как и другие паттерны, MVVM предлагает решение общих проблем. При правильной реализации он способен упростить разработку приложений для Windows 8. К сожалению, его неправильная реализация ведет к замедлению работы над проектами и неоправданному их усложнению. С помощью паттерна MVVM я создал десятки серьезных промышленных приложений, и счастлив, что компания Microsoft решила сделать его частью встроенных шаблонов приложений для Windows 8. Впервые паттерн MVVM появился в Visual Studio гораздо раньше, в панорамных шаблонах, применяемых в разработке для платформы Windows Phone. Хотя данный паттерн используется уже многие годы, существует много связанных с ним заблуждений. В табл. 9.1 приведен список характерных ошибок и разъяснено реальное положение дел.

В оставшейся части главы вы узнаете о различных частях MVVM и о том, как применять этот паттерн с особым упором на тестирование. Если вы прочли предыдущую главу и разбирали приведенные в ней примеры, понимание MVVM у вас уже есть, поскольку вы создавали специальные классы, реализующие уведомления об изменениях свойств для системы привязки данных. Эти классы на самом деле можно считать моделями

представления. Подробнее о моделях представления мы поговорим далее в этой главе.

Таблица 9.1. Типичные заблуждения, касающиеся паттерна MVVM

Неверно	Верно
Паттерн MVVM очень сложен	Применение паттерна MVVM может быть предельно простым при его верной реализации
С паттерном MVVM нельзя использовать вспомогательный код	Вспомогательный код (code-behind) — это просто расширение декларативной XAML-разметки представления. Представление ответственно за управление пользовательским интерфейсом, в том числе за пользовательский ввод, поэтому нет причин, по которым вспомогательный код не может работать с событиями и взаимодействиями
Паттерн MVVM сложен в реализации	Существует множество платформ (например, MVVM Light и Caliburn.Micro), которые позволяют за считанные минуты создать и запустить проект. Это относится и к встроенным в Visual Studio 2012 шаблонам для C# с XAML-разметкой
Паттерн MVVM делает ненужными конвертеры значений	Конвертеры значений — это многократно используемые, тестируемые фрагменты кода, способные отображать данные модели на представление, поэтому нет причин, по которым нельзя использовать конвертеры вместе с MVVM
MVVM снижает производительность приложения	Неправильная реализация любого паттерна пагубна для производительности. Правильная реализация паттерна MVVM упрощает модульное тестирование, что может помочь в тонкой настройке приложения и повышении производительности
MVVM годится лишь для очень больших проектов	Хорошая основа в виде MVVM вместе с твердым пониманием этого паттерна подходит как для маленьких проектов, так и для больших
MVVM касается лишь систем команд и сообщений	MVVM лишь распределяет роли между различными модулями внутри кода. Системы команд, сообщений и другие конструкции — это исключительно помощники и строительные блоки
В MVVM тяжело разобраться	MVVM не сложнее для понимания, чем система привязки данных или диспетчер визуальных состояний, поскольку на самом деле данный паттерн — это всего лишь описание того, как наилучшим образом использовать эти инструменты

Теперь, когда мы разобрались с заблуждениями по поводу MVVM, поговорим о достоинствах этого паттерна. А их немало, даже если не брать в расчет разделение рабочих процессов дизайнеров и программистов. На основе собственного опыта я могу привести список из десяти главных достоинств паттерна MVVM при его применении в приложениях.

1. **Четкое разделение ролей (ослабление связанности системы).** MVVM соответствует передовому опыту в отношении архитектуры программного обеспечения.
2. **Рабочие процессы дизайнеров и программистов.** MVVM позволяет многочисленным сотрудникам команд дизайнеров и программистов работать над одним проектом параллельно. Это достигается за счет поддержки данных, предназначенных исключительно для дизайнеров.
3. **Модульное тестирование.** Подробно о тестировании мы поговорим далее в этой главе.
4. **Привязка данных.** MVVM непосредственно использует все богатство и мощь системы привязки данных в XAML-разметке приложений для Windows 8, что также способствует поддержке данных, предназначенных для дизайнеров.
5. **Совершенствование механизма многократного использования кода.** Модели представления можно применять для усиления возможностей множества представлений, а различные вспомогательные функции и процедуры можно многократно использовать как в вашем проекте, так и в других программных продуктах вашей организации и даже в других технологиях, в том числе WPF и Silverlight.
6. **Модульность.** MVVM способствует модульному подходу к созданию дизайна, что упрощает модификацию отдельных частей приложения независимо от остальных.
7. **Сокращение объема переработки кода.** Благодаря четкому разделению ролей, MVVM минимизирует влияние, которое оказывает переделка одной части кода приложения на другие.
8. **Расширяемость.** Хорошо продуманный дизайн базовой структуры, каковой и является MVVM, упрощает расширение приложения за счет добавления новых экранов и модулей.
9. **Инструментальная поддержка.** Различные инструменты, такие как Expression Blend и визуальный конструктор, встраиваются в Visual Studio, что позволяет непосредственно использовать возможности MVVM.
10. **Личный запас паттернов.**

Последний пункт этого списка, «личный запас паттернов», нуждается в дополнительном пояснении. Когда вы учитесь читать, очевидна взаимосвязь

между вашим словарным запасом и способностью понимать прочитанное, так как словарный запас — это те строительные блоки, из которых строится текст, и неумение обращаться с этими блоками ведет к ошибочным заключениям и неправильной интерпретации текста. Однако сам по себе словарный запас еще не гарантирует понимание, так как вы должны уметь соединять слова друг с другом, чтобы понимать смысл текста в целом, и здесь существует взаимосвязь.

Разработка программного обеспечения также требует определенного словарного запаса. А начинаете вы со словарного запаса того языка, на котором программируете. У программ есть собственные синтаксис и грамматика, и их понимание основывается на вашей способности верно интерпретировать ключевые слова и понимать их в контексте программы.

Паттерны предлагают свой высокоуровневый словарный запас, позволяющий описывать целые подсистемы и компоненты в системе. Как и в случае с обычным словарным запасом, знать паттерн — это не то же самое, что знать, как его лучше приспособить к конкретному приложению (или можно ли вообще приспособить).

Чем лучше вы способны понимать и встраивать паттерны, тем шире ваши возможности в плане построения словарного запаса и тем лучше вы понимаете сложные программные системы. Я обнаружил, что программисты, которые участвуют в самых успешных проектах, работают с самыми сложными системами, обычно обладают весьма серьезным личным запасом паттернов. Они не только знают о многих паттернах, существующих в сфере разработки программного обеспечения, но и понимают, когда и где их лучше всего применять.

Я думаю, что паттерн MVVM так популярен потому, что при его верной реализации он позволяет получить все те преимущества, о которых я рассказал. MVVM — это очень важный паттерн, который нужно изучать и понимать, чтобы успешно разрабатывать приложения для Windows 8, поскольку по умолчанию именно он используется во встроенных XAML-шаблонах приложений на C#. Но, как и все паттерны, это лишь инструмент, и с ним нужно уметь правильно работать. В следующих нескольких разделах я подробно расскажу о паттерне MVVM. Это должно помочь вам изучить этот паттерн, чтобы вы знали, как максимально эффективно применять его в приложениях. А начнем мы с обзора компонентов, составляющих MVVM.

Модель

Понятие модели часто путают с более узким понятием модели данных. Гораздо лучше здесь подходит определение этого понятия как «модель мира» для приложения. Это та модель, которая охватывает все, что должно

происходить при решении бизнес-проблем приложения *без* определения конкретного пользовательского интерфейса или уровня представления данных. Некоторым нравится называть ее *моделью предметной области* (domain model), но модель предметной области — это концептуальное понятие, в то время как *модель* в MVVM представляет собой реальную реализацию.

Вот простой пример. Банковая система удаленного обслуживания клиентов может содержать сведения о клиентах и счетах. Представление клиентов и счетов — это часть модели. Модель описывает их взаимосвязь. У клиента может быть один или больше счетов. Модель описывает состояния (счет может быть открыт или закрыт) и позволяет выполнять некоторые действия (на счетах накапливаются проценты). Для того чтобы модель работала, нужна реализация некоторых классов со свойствами. Нужна база данных для хранения информации. Нужны прикладные программные интерфейсы для получения данных, их передачи и применения к ним различных алгоритмов обработки.

Правильно построенная модель обеспечивает доступ лишь к тем ее частям, которые нужны приложению. Например, уровню представления не нужно заботиться о том, как хранятся данные (в базе данных или в XML-файле), или о том, как они извлекаются из хранилища (обрабатываются и отправляются в виде двоичного объекта с помощью сокета TCP/IP или передаются с использованием REST-службы). Модель, предоставляющая слишком много данных, порождает ненужные зависимости и чрезмерно усложняет код.

Модель для приложения Wintellog объединяет блоги и посты блогов, а также сетевые и транслирующие прикладные программные интерфейсы, чтобы их получать. Кроме того, она использует прикладные программные интерфейсы среды WinRT для обновления плиток и отправки уведомлений. Код, применяемый для сохранения позиций в кэше и загрузки их оттуда, также является частью модели. Все эти компоненты работают вместе, чтобы предоставить пользователю все необходимые данные, но напрямую с пользователем они не взаимодействуют. Когда в приложении выводится на экран та или иная позиция блога, это не физическая страница в Интернете, а представление данных сущности `BlogItem`.

Ваша основная цель должна заключаться в том, чтобы писать гибкий, расширяемый код, который хорошо поддается тестированию и поддержке. Если модель вашего приложения соответствует этим принципам, паттерн MVVM можно легко связать с необходимыми интерфейсами и классами, не образуя зависимостей с теми частями системы, которые не имеют ничего общего с логикой уровня представления. Модель — это «модель реального мира» для приложения, но в определенный момент ее нужно представить конечному пользователю. Это достигается посредством вывода данных, что в случае с приложениями для Windows 8 реализуется мощным

пользовательским интерфейсом, обладающим богатыми возможностями. То, что видит пользователь, и называется *представлением* (view).

Представление

Представление в приложениях для Windows 8 — это та их часть, которая взаимодействует с пользователем. Представление само по себе является пользовательским интерфейсом. Обычно для его создания применятся декларативная XAML-разметка. XAML участвует в системе зависимых свойств. Представление может как выводить некоторую информацию для пользователя, так и реагировать на ввод данных. В табл. 9.2 показаны обычные составляющие представления и описаны их функции.

Таблица 9.2. Представление в MVVM

Компонент	Описание
XAML	Декларативная разметка для описания макета, элементов управления и других объектов, формирующих пользовательский интерфейс
Конвертеры значений	Специальные классы, которые служат для преобразования данных в типы, подходящие для пользовательских элементов данных, и обратно
Шаблоны данных	Шаблоны, которые отображают элементы данных на элементы управления
Группы визуальных состояний	Именованные состояния, которые воздействуют на свойства различных элементов, чтобы получить физическое состояние на основе логических состояний элементов управления
Раскадровки	Анимации и переходы
Режимы работы	Многократно используемые алгоритмы, которые можно применять к различным элементам управления, что обычно делается с помощью присоединенных свойств
Триггеры	Алгоритмы, которые можно применять к элементам управления и вызвать посредством сконфигурированных событий, что обычно организуется с помощью присоединенных свойств
Вспомогательный код (code-behind)	Расширения XAML-разметки, предназначенные для решения дополнительных задач, относящихся к пользовательскому интерфейсу

Судя по таблице, должно быть очевидно, что представление нельзя назвать полностью изолированным от логики уровня представления. Команды

в модели представления отображают элементы управления на действия, объявления привязки данных требуют для привязки знания структуры нижележащих данных. Анимации, визуальные состояния, шаблоны, режимы работы и триггеры отражают различные компоненты бизнес-логики, которые имеют отношение к представлению.

Здесь не вполне очевидно то, что все эти компоненты не сохраняют состояния, относящегося к модели приложения. Раскадровки поддерживают состояние (запущена, остановлена, воспроизводится и т. д.), группы визуальных состояний поддерживают состояние, но только для пользовательского интерфейса. Режимы работы и триггеры также действуют на основе событий или базовых элементов управления и не должны зависеть от нижележащих данных или бизнес-логики. Даже вспомогательный код обычно пишут лишь для поддержки определенных аспектов пользовательского интерфейса. Более сложный код должен располагаться где-то еще, и не потому, что существует правило, согласно которому вспомогательный код запрещен, а из-за того, что более сложные алгоритмы нужно тестировать. Когда подобный алгоритм выделен в отдельный изолированный класс, это упрощает тестирование, поскольку позволяет не возиться сразу со всем пользовательским интерфейсом.

Итак, где же находится основная часть логики уровня представления и что отвечает за состояние бизнес-логики приложения? В это состояние должны входить данные, показываемые пользователю, а также статус различных команд и процессов, которые управляют пользовательским интерфейсом и обеспечивают реакцию на пользовательский ввод. Ответ заключается в важнейшей части паттерна MVVM, в том самом элементе, который делает этот паттерн уникальным, — в модели представления.

Модель представления

Именно *модель представления* (view model) придает уникальность паттерну MVVM. Это обычный класс, который отвечает за координацию взаимодействия представления и модели. В модели представления должна располагаться основная часть логики уровня представления. На мой взгляд, хорошо написанную модель представления можно тестировать, вообще не создавая никаких представлений. Существует три основных метода взаимодействия с представлением:

- Привязка данных.
- Визуальные состояния.
- Вызовы методов и/или команд.

Учитывая это, можно сказать, что мы уже создали несколько моделей представления. Класс `BlogDataSource` предоставляет объект `GroupLists`, который, в свою очередь, содержит экземпляры классов `BlogGroup` и `BlogItem`. Модели представления обычно реализуют интерфейс, предусматривающий уведомления об изменениях свойств. Часто они содержат ссылки на прикладные программные интерфейсы, позволяющие обмениваться данными и взаимодействовать с моделью. Наличие интерфейсов упрощает написание переносимого кода, который можно многократно использовать в разных проектах вашей организации.

Все это — компоненты паттерна MVVM. Модель представления играет ключевую роль в MVVM, она является классом, который поддерживает уведомления об изменениях свойств, то есть может участвовать в привязке данных. Модель представления может быть достаточно простым объектом, наподобие экземпляра класса `BlogItem`, а может быть и классом, который представляет коллекции, команды и взаимодействует с другими интерфейсами, что больше похоже на объект `BlogDataSource`. Разделение модели представления и пользовательского интерфейса не только позволяет осуществлять тестирование, но и способствует многократному использованию кода. В комбинации с переносимой библиотекой классов вы сможете эффективно писать бизнес-логику для различных платформ.

Переносимая библиотека классов

Переносимая библиотека классов (Portable Class Library, PCL) — особый тип проекта в Visual Studio 2012, который позволяет создавать сборки, способные работать на множестве .NET-платформ. Это идеальный инструмент создания компонентов, содержащих бизнес-логику, которые можно многократно использовать как в настольных приложениях, так и в приложениях для Windows 8. Сборка, построенная на основе шаблона PCL-проекта, может исполняться на этих платформах без перекомпиляции.

PCL позволяет указать набор целевых прикладных программных интерфейсов, общих для всех платформ. При разработке PCL-проекта вам предлагается выбрать платформы, на которых вам хотелось бы запускать свою сборку (рис. 9.2).

То, какой именно объем переносимого кода будет доступен, зависит от того, насколько устарели выбранные целевые платформы, от их количества и типа. Например, проекту для Xbox будет доступно гораздо меньше прикладных программных интерфейсов, чем проекту для Silverlight.

Аналогично, платформа .NET Framework 4.0 накладывает больше ограничений на проект, чем .NET Framework 4.5. Для того чтобы увидеть список доступных прикладных программных интерфейсов, разверните ветвь References (Ссылки) вашего проекта в обозревателе решений, щелкните правой кнопкой мыши на интересующей вас ссылке и выберите в появившемся меню команду Properties (Свойства). Вы увидите окно, в котором есть свойство Path (Путь). Скопируйте путь из этого свойства и откройте его в проводнике Windows.

Рис. 9.2. Выбор целевой платформы для переносимой библиотеки классов

На рис. 9.3 показана папка, которая соответствует проекту, созданному с сохранением значений, установленных по умолчанию. Каждая комбинация платформ приводит к созданию нового профиля, объединяющего набор прикладных программных интерфейсов, способных работать на всех выбранных платформах. При этом используется новый инструмент, который называется расширением SDK (Extension SDK). Этот инструмент позволяет задействовать одну ссылку для обращения одновременно к множеству файлов и конфигураций, а не на единственный проект или сборку. Больше о расширении SDK вы можете узнать на странице http://msdn.microsoft.com/ru-ru/library/hh768146(v=vs.110).aspx.

Как видите, профиль нашего примера поддерживает несколько библиотек. В дополнение к основным сервисам он обеспечивает работу сети и веб-служб, сериализацию, а также XML, включая LINQ. Для исследования прикладных программных интерфейсов, доступных в сборках, на которые ссылается проект, вы можете использовать утилиту ILDASM.EXE. В данном случае сборки — это просто копии соответствующих сборок для .NET Framework, поддерживаемые конкретным профилем.

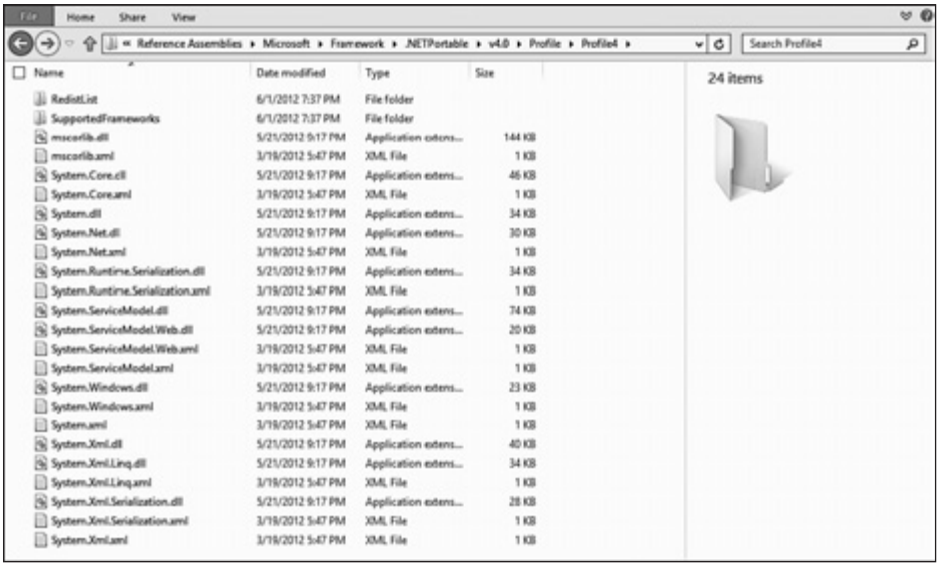


Рис. 9.3. Профиль переносимой библиотеки классов

Учебный проект WintellogMvvm, относящийся к главе 9, был переработан для использования переносимой библиотеки классов. Проект PortableWintellog создавался так, чтобы поддерживать и .NET Framework 4.5, и приложения для Windows 8. Данный проект позволяет работать с множеством классов и интерфейсов, которые без изменений можно применять как в настольных программах, так и в приложениях для Windows 8.

Папка Contracts содержит интерфейсы, которые совместно используются различными платформами. PCL-проект — это отличное место, где можно определить интерфейсы и контракты, строго не зависящие от непереносимых прикладных программных интерфейсов. Благодаря концепции, которая называется *инверсией управления* (Inversion of Control, IoC), эти интерфейсы помогут вам отделить логику, которая специфичная для конкретной платформы, от другой логики. IoC способствует созданию классов, которые проще тестировать и которые с высокой вероятностью можно будет использовать на различных целевых платформах.

В предыдущей версии приложения класс `StorageUtility` был статическим классом, который предназначался для работы с локальным кэшем приложения. Доступом к `StorageUtility` управлял класс `BlogDataSource`. Это означало, что на него возлагалась дополнительная ответственность за взаимодействие со статическими методами, предоставляемыми классом `StorageUtility`. Хотя данный подход позволял весьма просто получать доступ к хранилищу, он порождал сильную зависимость между источником данных и реализацией логики хранения, что мешало совместному исполь-

зованию класса `BlogDataSource` на платформах, отличных от Windows 8. Кроме того, это усложняло тестирование класса, так как любой тест требовал подходящего хранилища.

В проекте PortableWintellog система контроля была инвертирована. Это всего лишь означает, что класс `BlogDataSource` больше не несет ответственности за хранилище. Вместо этого он работает с интерфейсом `IStorageUtility`. Интерфейс предоставляет сигнатуры методов для сохранения и восстановления позиций, но не предусматривает реализации методов. Это позволяет ослабить связь с классом `BlogDataSource`, так как исключается прямая зависимость от конкретной реализации системы хранения.

У такого подхода есть масса преимуществ. Во-первых, можно очень просто тестировать логику хранения в классе `BlogDataSource` без использования самого хранилища. В целях тестирования можно легко создать вспомогательный класс (как вы увидите далее в этой главе) для имитации работы с хранилищем. Во-вторых, вы можете реализовать хранилище, подходящее для конкретной целевой среды исполнения приложения. Проект WintellogMvvm содержит реализацию хранилища данных, рассчитанную на приложения для Windows 8. Проект WintellogWpf представляет собой настольное WPF-приложение, содержащее реализацию хранилища для настольных приложений. Несмотря на то что один и тот же механизм реализован по-разному, интерфейс позволяет использовать в обеих версиях приложения один и тот же класс `BlogDataSource`.

Взгляните на проект WintellogWpf. Он ссылается на переносимую библиотеку классов и многократно использует логику, представляемую классом `BlogDataSource`. Сюда относится логика загрузки отдельных блогов и позиций в списке, взаимодействие с кэшем и даже некоторые онлайновые функции. Класс `HttpClient` совместно используется и в настольной версии приложения, и в версии для Windows 8, поэтому он согласованно загружает страницу поста и обрабатывает связанные с ней изображения.

Реализация `StorageUtility` задействует для хранения кэша локальную файловую систему, а не изолированное хранилище. Для определения пути к папке приложения используется внутренний вспомогательный метод:

```
private static string GetRootPath()
{
  return Path.Combine(
    Environment.GetFolderPath(
      Environment.SpecialFolder.LocalApplicationData),
"Wintellog");
}
```

Далее показан код для просмотра файлов. Прежде чем запросить список файлов, этот код проверяет, существует ли соответствующая папка. Код заключен в оболочку `Task` для асинхронного выполнения:

```
var directory = Path.Combine(GetRootPath(), folderName);
return
  Directory.Exists(directory)
    ? Directory.GetFiles(directory)
    : new string[0];
```

Помните об упомянутой ранее инверсии управления? Приложение управляет зависимостями, внедряя их при запуске. В файле App.xaml.cs вы найдете следующие фрагменты кода, которые отображают переносимые контракты на их WPF-реализации:

```
Ioc = new TinyIoc();
Ioc.Register<IStorageUtility>(ioc => new StorageUtility());
Ioc.Register<IApplicationContext>(ioc => new ApplicationContext());
Ioc.Register<IDialog>(ioc => new WpfDialog());
Ioc.Register<ISyndicationHelper>(ioc => new SyndicationHelper());
```

После разрешения всех зависимостей вспомогательные IoC-механизмы используются для внедрения их в класс `BlogDataSource` посредством его конструктора:

```
Ioc.Register(ioc => new BlogDataSource(
ioc.Resolve<IStorageUtility>(),
ioc.Resolve<IApplicationContext>(),
ioc.Resolve<IDialog>(),
ioc.Resolve<ISyndicationHelper>()));
```

Подобный прием называется внедрением зависимостей (Dependency Injection, DI) — это обычная методика, применяемая при реализации IoC. Небольшой реализующий возможности IoC класс, который включен в учебное приложение, — эти лишь простейшая демонстрация сценариев, доступных благодаря этому механизму. Существует немало серьезных платформ, в том числе Managed Extensibility Framework (MEF). Ее можно использовать в приложениях для Windows 8 посредством NuGet-пакета (http://nuget.org/packages/Microsoft.Composition/).

Когда вся логика инкапсулирована в различные совместно используемые и локальные классы, весьма просто создать в файле MainPage.xaml XAML-разметку для блогов. Здесь реализована более простая функциональность, чем в приложении для Windows 8, но она полностью задействует

существующие классы. Фактически здесь показано, как получать приложения, значительно различающиеся по внешнему виду и режиму работы, но реализующие в основных классах одну и ту же бизнес-логику. Сказанное иллюстрирует рис. 9.4.

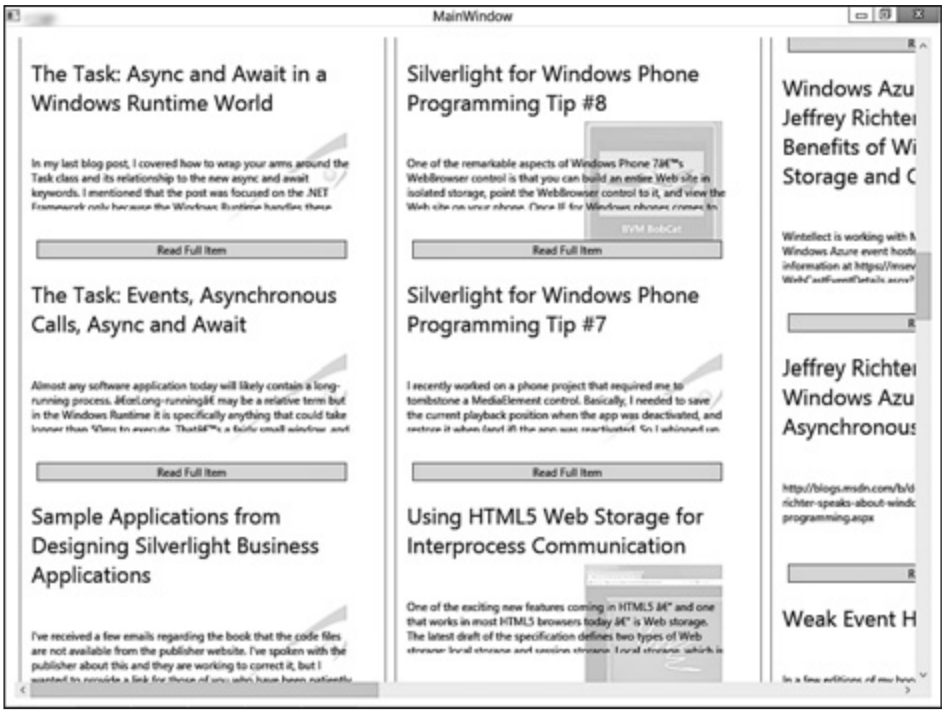


Рис. 9.4. WPF-версия приложения

Перенос кода между различным платформами и приложениями — это очень хорошо. Однако гораздо более серьезное преимущество, которое получает обычный программный код благодаря переносимой библиотеке классов, заключается в тестировании. Тестировать основные компоненты приходится лишь раз, хотя после тестирования использовать их можно будет в различных приложениях. И здесь у вас может появиться вопрос: «А в чем смысл тестирования программного обеспечения?»

Зачем тестировать приложения?

«Зачем тестировать приложения?» — это первый вопрос, который мне обычно задают, когда я говорю о тестировании.

Я создаю корпоративное программное обеспечение в течение более десяти лет, поэтому обычно воспринимаю тестирование как должное. На самом деле это очень хороший вопрос, как и другие вопросы, которые мне часто задают, когда компании планируют выделение ресурсов и создание команды разработчиков: «Если приложение хорошо написано, на самом ли деле оно нуждается в тестировании?»; «Я понимаю, что тестирование нам нужно, но не разработчик ли должен его выполнять?»; «Модульное тестирование только замедляет работу — может быть, нам достаточно менеджера, который проверит приложение, когда оно будет готово?»

Есть десятки веских доводов в пользу тестирования. И я собираюсь рассказать о некоторых достоинствах тестирования, выявленных мною за годы работы. Эти достоинства не зависят ни от структуры, ни от размера проекта, ни от состава команды, которая над ним работает.

Тестирование позволяет избежать допущений

Разработчики не объективны. Невозможно отрицать, что когда вы начинаете создавать приложение, у вас есть представление о том, как оно должно работать. Если это представление верно, вы легко можете разработать приложение, которое действительно нужно пользователям. Однако обычно это не так, и большинство разработчиков создают то, что нравится им. Вы можете целыми днями тестировать свое приложение и не найти ни единой ошибки. Позвольте пользователям поработать с продуктом, и все будет выглядеть уже не столь радужно. Ведь пользователи не знают, что поля для чисел не предназначены для ввода текста, а для ввода дат нужно использовать особый формат; они не знают, что при работе с последовательностью взаимосвязанных форм не следует закрывать браузер или отключать ноутбук от зарядного устройства.

Пользователи неизменно придумывают такие способы взаимодействия с приложениями, которые оказываются совершенно неожиданными для разработчиков, а это приводит к ошибкам. И наша задача как разработчиков — предусмотреть подобные сценарии развития событий. Это подразумевает не только поиск ошибок в коде и их протоколирование, но и работу с командой дизайнеров над созданием такого пользовательского интерфейса, который делает совершенно очевидным, для чего предназначена программа и как с ней работать. Связать все воедино — большое искусство, и как мне говорил один дальновидный руководитель: «Для того чтобы создать иллюзию простоты, требуется немало усилий».

Спецификация конкретной программы или модуля часто фокусируется на вероятных событиях, а также на каких-то простых проверках данных. Каждый шаг в процессе разработки сопровождается допущениями о макете, о соответствующей проверке, о бизнес-логике. Тестирование, особенно тогда, когда им занимается команда, члены которой представляют собой потенциальных пользователей приложения, причем не являющихся техническими специалистами, помогает на самых ранних этапах разработки исключать допущения, что в результате может привести к созданию высококачественного продукта.

Тестирование убивает ошибки на корню

Если кто-то попросит вас помочь ему найти часы, которые он потерял пять лет назад, с чего вы начнете? Представьте, сколько людей за это время могло пройти там, где они были потеряны. Часы могло сдуть сильным ветром. На том месте могло начаться строительство, и часы оказались под кучей земли. Однако если тот же самый человек попросит вас помочь ему найти часы через пять минут после пропажи, вы, возможно, будете больше уверены в успехе поисков.

Чем ближе найденные ошибки к источнику, тем легче их исправлять. Когда вы изолируете ошибку внутри конкретного компонента, вам проще понять окружение, в котором выявлена ошибка, и предпринять шаги для решения проблемы. Гораздо сложнее находить ошибки, возникающие в результате взаимодействия сложных модулей, в которое к тому же вовлечены компоненты сторонних производителей. Ошибки, выявленные в ходе практической работы с программой, часто еще сложнее идентифицировать. Дело в том, что почти невозможно получить точное описание, которое необходимо для правильной идентификации проблемы в производственной среде.

Тесты помогают находить ошибки на ранних этапах разработки. Чем раньше вы обнаружите ошибку, тем легче будет ее исправить. Модульные тесты позволяют проверить правильность функционирования компонентов на локальном уровне. Интегральные тесты позволяют убедиться в том, что модули функционируют верно в условиях сопутствующих изменений в нижележащем коде или в прикладных программных интерфейсах. Кодовые тесты пользовательского интерфейса позволяют оценивать правильность работы интерфейса и находить несоответствия, которые меняют порядок работы с приложением и часто являются побочным эффектом изменения бизнес-логики приложения или механизмов проверки правильности данных.

Тестирование помогает документировать программный код

Мне особенно нравятся проекты с открытым кодом, которые включают в себя тысячи строк отлично написанных модульных тестов. Для меня это самый простой и быстрый способ понять, как работать с API стороннего производителя. Разобраться в этом можно и самостоятельно, но в тестах есть примеры кода, множество вариантов вызовов API и, что гораздо важнее, условия, которые способны привести к ошибке. Я изучал множество проектов, и далеко не всегда мне по документации удавалось понять, как использовать компоненты, зато я понимал это с помощью тестов.

Хорошо написанные модульные тесты с адекватным охватом кода помогают документировать программы. Благодаря им разработчикам проще изучать, поддерживать и расширять код. Модульные тесты дают возможность заглянуть вглубь проекта. Они иллюстрируют различные способы вызова прикладных программных интерфейсов. Они показывают условия, в которых могут происходить исключения, и то, как обрабатывать эти исключения. Они предоставляют основу для дальнейшей работы, если вы расширяете код, и служат шаблоном для дополнительных тестов при развитии базы программного кода проектов.

Интегральные тесты и кодовые тесты пользовательского интерфейса также требуют особого сценария. Этот сценарий подразумевает знание системы и того, как она предположительно функционирует. Довольно часто варианты тестов пишутся на начальных этапах проекта как часть процесса подготовки спецификации. В качестве разработчика я всегда предпочитаю иметь точные хорошо продуманные варианты тестов, так как они дают мне возможность оценить результаты собственной работы. Алгоритм работы приложения — это прекрасно, и я могу писать код в соответствии с ним, однако набор точных тестов — это нечто вроде контрольного списка, который я могу использовать, чтобы оценить правильность написания кода и правильность моего понимания алгоритма в контексте бизнес-цели проекта.

Тестирование упрощает расширение и поддержку приложения

Этот пункт вполне согласуется с предыдущим. Я до сих пор помню один проект, которым занимался в самом начале карьеры. Мне пришлось начать писать модульные тесты для него, а я в то время не вполне понимал их значение. Стиснув зубы, я писал один тест за другим, полагая в душе, что бесполезно трачу время, отвлекаясь от главного. И хотя некоторые из тестов

были написаны не очень хорошо, благодаря тому что наш руководитель обладал прекрасным даром убеждения, большинство из них, к счастью, достойно выполняли свои функции.

А то, насколько достойно, я понял лишь тогда, когда мне предложили внести серьезное изменение в один из интерфейсов, потребовавшее радикальной переделки кода. В прошлом такие упражнения больше напоминали программирование вслепую. Я завершил модернизацию, вдохнул поглубже, надеясь, что код скомпилируется, и затем в пошаговом режиме запустил готовый продукт, скрестив пальцы. Мне снова пришлось затаить дыхание, когда приложение пошло в производство, поскольку всегда вдруг возникала неожиданная проблема и, как результат, — забавные утренние звонки в службу поддержки.

С модульными тестами все изменилось. Я получил возможность переделывать части интерфейса и затем проводить модульные тесты, проверяя, какие области подверглись влиянию. Вместо того чтобы воспринимать приложение как единое целое, пытаясь понять, что в нем разладилось, я мог сконцентрироваться на модульных тестах и определить то, что требовалось в каждый конкретный момент. Это позволило сделать изменения быстрыми, точными и воспроизводимыми. В результате не только вдвое сократилось время, необходимое на переделку, но и готовый продукт получался более надежным.

Тестирование улучшает архитектуру и дизайн

Один из побочных эффектов написания тестов заключается в том, что это заставляет вас задуматься, как построены и как взаимодействуют компоненты. И этот позитивный побочный эффект приводит к улучшению архитектуры приложения и подхода к разработке его дизайна. Я столкнулся с этим сам, когда мы вместе с большой командой перешли от спецификаций, «написанных на салфетках», к процессу, подразумевающему написание полноценных модульных тестов. Нередко программисты запускают вместе классы и прикладные программные интерфейсы только для того, чтобы понять, что они должным образом не работают и требуют переделки. Тесты помогают выявлять проблемы с прикладными программными интерфейсами уже на ранних этапах разработки, поэтому к тому времени, когда их нужно интегрировать с другими компонентами, они оказываются доведенными до совершенства и проверенными.

Я столкнулся с этим в начале карьеры. Создавая сложные системы, я часто бросался с головой в глубины программирования и выныривал оттуда, только написав тысячи срок кода. Единственная проблема заключалась

в том, что мало было просто щелкнуть выключателем, чтобы быстро отыскать даже небольшой изъян в программе, для этого требовалась куча времени, а чтобы исправить — и того больше. Когда я начал использовать тесты, я начал воспринимать большие и сложные системы как маленькие простые модули. Я мог полностью концентрироваться на дизайне и конструкции, на тщательном тестировании компонента, после чего мог двигаться дальше. Для создания таких вот строительных блоков системы требуется чуть больше времени, зато они быстро и легко интегрируются и экономят время в целом, поскольку сокращают количество ошибок в готовом проекте.

Тестирование повышает культуру разработчиков

Я уверен, что тестирование оказывает такое же положительное влияние на программистов, как и на качество кода. Вот еще одно наблюдение из моего опыта. Те самые разработчики, которые в начале проекта ворчали, когда им приходилось писать тесты, часто с удовольствием делились с другими своими модулями и предлагали тесты для их проверки. Трата дополнительного времени на обдумывание компонентов и подготовку тестов полностью меняет подход к разработке программного обеспечения. Благодаря положительному влиянию тестов можно быстро и легко находить общие паттерны и применять их в других местах. То же касается обработки ошибок, определения API и организации алгоритмов.

Вывод: пишите модульные тесты!

Надеюсь, теперь и вы согласны с тем, что модульные тесты — это очень важно. Некоторые разработчики, читающие эти строки, могут даже удивиться, что приходится об этом напоминать. Существуют многие организации, которые де-факто считают тесты неотъемлемой частью разработки, другие полностью следуют методике разработки через тестирование (test-driven development). Если вы новичок в тестировании, я советую вам осваивать его поэтапно. И лучше всего начать со включения в ваши проекты модульных тестов.

Модульные тесты

Модульные тесты относятся к категории тестов, известных как тесты *белого ящика* (white box). Программную систему можно рассматривать с двух точек зрения. Конечный пользователь воспринимает ее как «черный ящик»,

поскольку он не видит исходного кода, определений баз данных и других механизмов, благодаря которым работает приложение. Однако он знает, что с системой можно взаимодействовать (обычно посредством пользовательского интерфейса): в черный ящик направляются команды, а из черного ящика появляются результаты. Все это относится к обеспечению качества и автоматизированным тестам пользовательского интерфейса.

Тесты белого ящика правильнее было бы называть тестами «прозрачного ящика», так как они подразумевают изучение внутренних механизмов системы. Модульные тесты требуют глубоких знаний того, что происходит в исходном коде и на уровне сохранных процедур. Интегральные тесты требуют знания прикладных программных интерфейсов и границ между системами. Тесты белого ящика гарантируют работоспособность системы изнутри, являясь тем самым набором «издержек и противовесов», который обеспечивает правильное функционирование черного ящика.

Я встречал множество тестов, которые хотя и назывались модульными, на самом деле таковыми не были. Если вы ненавидите проводить для вашего проекта тесты, потому что они требуют подключения к Интернету, особым образом настроенного каталога и длятся долгие часы, вы, вероятнее всего, запускаете интегральные тесты, а не модульные. У модульных тестов есть некоторые общие черты, которые важно понимать.

- Модульные тесты должны фокусироваться на конкретной функции класса. На класс следует возлагать не более одной задачи, и если вы тратите много часов на написание модульных тестов для класса, вероятнее всего это может указывать на то, что класс нуждается в переделке.
- Модульные тесты должны быть полностью изолированными, чтобы их можно было проводить, «несмотря ни на что». Не следует создавать для них какую-то особую среду, они не должны зависеть от других классов, не относящихся с тому, который вы тестируете.
- Модульные тесты должны быть короткими и простыми. Их проведение не должно отнимать много времени, даже если в вашем проекте их много. Должно быть предельно понятно, что делает тест и как интерпретировать его результаты.
- Модульные тесты должно быть легко писать. По моему мнению, самый простой и эффективный способ для написания тестов — создавать их при определении целевого класса. Методика разработки через тестирование (Test-Driven Development, TDD) идет даже дальше, предлагая *сначала* писать тест, а потом уже определять класс и API, которые можно будет проверить с помощью этого теста.

Подробности о методике разработки через тестирование вы можете узнать на странице http://www.agiledata.org/essays/tdd.html.

Я создал несколько модульных тестов для приложения Wintellog, чтобы вы быстрее разобрались с ними. Если бы я делал это, полностью согласуясь с методикой разработки через тестирование, потребовались бы буквально сотни тестов. Однако для целей данной главы я ограничился несколькими примерами. Все тесты, включенные в учебное приложение для главы 9, можно запустить с использованием среды модульного тестирования приложений для Магазина Windows (Windows Store Unit Testing Framework).

Среда модульного тестирования приложений для Магазина Windows

Visual Studio 2012 обеспечивает встроенную поддержку модульных тестов. Система, предлагаемая по умолчанию, известна как интегрированная среда MSTest. Хотя Visual Studio 2012 поддерживает подключение других сред тестирования, в том числе популярной среды NUnit (http://nunit.org/), я ограничился теми инструментами, которые доступны сразу же после установки профессиональной версии продукта (к несчастью, модульные тесты в бесплатной Express-версии Visual Studio 2012 недоступны). В частности, здесь имеются шаблоны модульных тестов приложений для Windows 8.

Когда вы добавляете в решение новый проект в стиле Windows 8, одна из доступных возможностей заключается в создании проекта на основе шаблона Unit Test Library for Windows Store apps (библиотека модульного тестирования приложений для Магазина Windows). При выборе этого шаблона будет создан проект, ссылающийся на сборки и SDK, применяемые для тестирования, в том числе на `MSTestFramework` и `TestPlatform`. Такой проект очень похож на проект полноценного приложения для Windows 8, у него даже есть собственный файл с манифестом. Пример такого проекта вы можете найти в учебном приложении для этой главы WintellogTest.

Для того чтобы провести тесты, выберите в меню команду Test ▸ Run ▸ All tests (Тест ▸ Выполнить ▸ Запустить все) или нажмите клавишу A, удерживая нажатыми клавиши Ctrl+R. Пакет будет скомпилирован и исполнен. На рис. 9.5 приведены результаты тестирования, в вашем случае результаты должны быть похожими.

В списке вы можете видеть сведения об отдельных тестах с пометками о том, прошел ли тест и сколько времени он занял. Этот список можно фильтровать с помощью меню фильтров, расположенном около кнопки поиска в верхней части окна. В дополнение к фильтрации по результатам вы

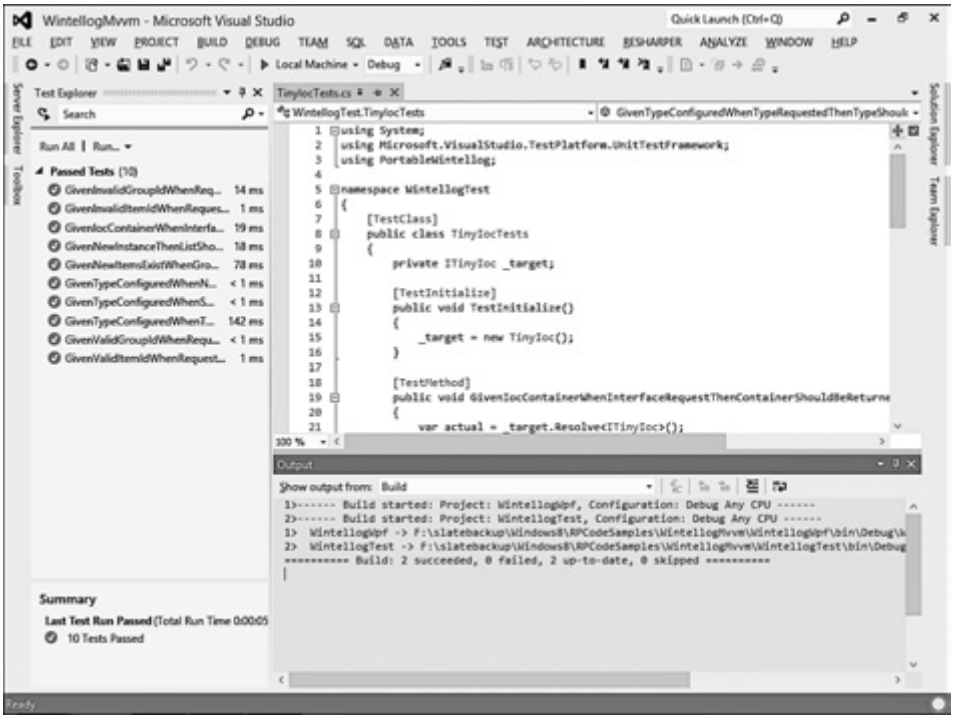


Рис. 9.5. Результаты модульного тестирования в Visual Studio 2012

можете проводить отбор тестов по именам файла и их полным именам. На рис. 9.6 с помощью фильтра отобраны только те тесты, которые находятся в файле TinyIocTests.cs.

Писать тесты довольно просто. Нужно объявить класс и пометить его атрибутом `TestClass`, например:

```
[TestClass]
public class TinyIocTests
```

При объявлении тестов используется множество популярных соглашений. Я предпочитаю, чтобы у меня было по одному файлу на каждый целевой класс, поэтому создаю тестовый класс и называю его именем целевого класса с приставкой `Tests`. Вот обычное описание конкретной цели, которую я отслеживаю в тесте:

```
private ITinyIoc _target;
```

Для любых конструкций, которые должны быть исполнены перед каждым отдельным тестом, вы можете объявить метод, пометив его атрибутом

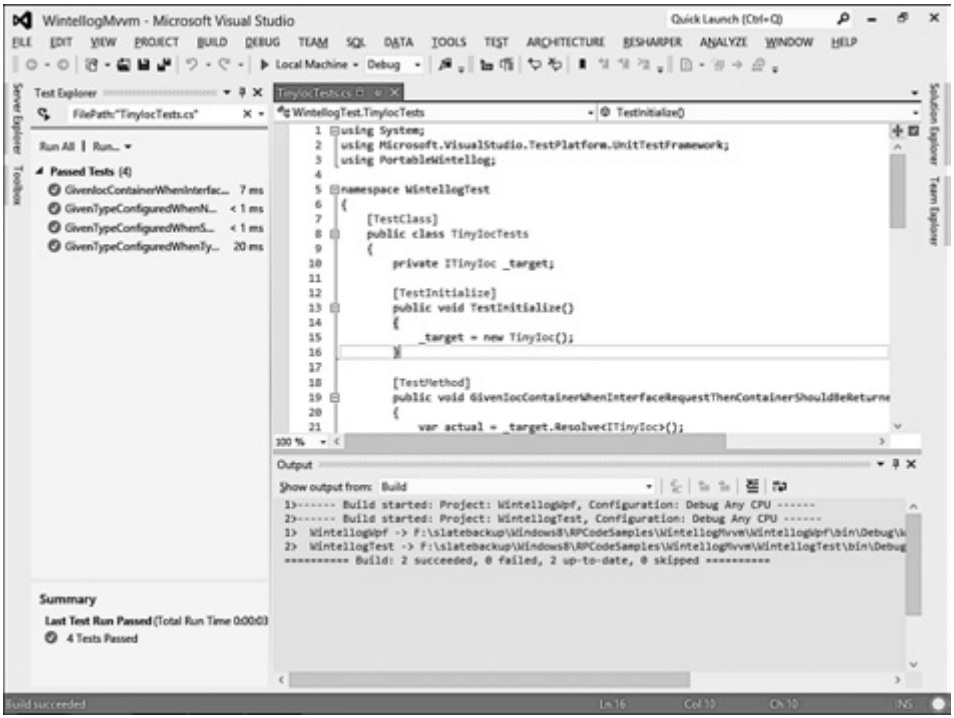


Рис. 9.6. Отбор тестов по файлу, в котором они описаны

`TestInitialize`. В этом примере просто создается новый экземпляр вспомогательного IoC-класса:

```
[TestInitialize]
public void TestInitialize()
{
  _target = new TinyIoc();
}
```

И наконец, тесты объявляются как методы, помечаемые атрибутом `TestMethod`. Есть множество подходов как к именованию тестов, так и к определению областей их видимости. Я стараюсь, чтобы целью тестов была отдельная подсистема или фрагмент кода в целевом методе, но иногда для этого требуются дополнительные тесты. Тесты мне нравится именовать в соответствии с эталоном *дано... когда... тогда*.

Дано — это определенный набор предварительных условий, *когда* — описание действий, которые нужно предпринять, *тогда* — описание ожидаемого результата. Хотя это помогает в именовании тестов, внутренняя структура теста практически всегда следует трем предопределенным шагам:

выставлению начальных условий и подготовке, проведению тестирования и сверке ожидаемых результатов с полученными. Вы *готовите* тест к исполнению (то есть указываете то, что «дано»), исполняете его, взаимодействуя с целевым классом (то есть переходите к части «когда»), и затем *сверяете* полученные результаты с ожидаемыми (то есть проводите оценку части «тогда»).

Начнем с простого теста. IoC-контейнер настраивается сам, поэтому если вы запрашиваете интерфейс `ITinyIoc`, он должен вернуть экземпляр IoC-контейнера, который вы создали. В соответствии с описанными принципами это можно описать так:

> *Дано*: создан вспомогательный IoC-объект; *когда* запрашивается IoC-интерфейс, *тогда* вспомогательный объект должен вернуть собственный экземпляр.

Код теста может выглядеть следующим образом:

```
[TestMethod]
public void GivenIocContainerWhenInterfaceRequest
   ↳ ThenContainerShouldBeReturned()
{
  var actual = _target.Resolve<ITinyIoc>();
  Assert.AreSame(_target, actual, "Test failed: instance was not
   ↳   returned.");
}
```

Подготовка происходит в методе `TestInitialize` при создании целевого объекта. *Действие* производится при вызове интерфейса. *Проверка* заключается в сравнении ожидаемого результата (возвращения объектом собственного экземпляра) с тем, что на самом деле было возвращено классом. Если обнаружится несоответствие, тест будет признан неудачным с выводом соответствующего сообщения. От этого простого примера мы можем перейти к более сложному:

> *Дано*: тип, сконфигурированный с помощью вспомогательного IoC-класса; *когда* осуществляется запрос к типу, *тогда* должен быть возвращен экземпляр соответствующего типа.

На самом деле это именно то, что должно происходить при инверсии управления. Простой контейнер, который я написал для этого приложения, позволяет передать нужный вам тип (обычно интерфейс или абстрактный класс, хотя это может быть и обычный класс) и функцию, возвращающую новый экземпляр типа. По умолчанию вспомогательный IoC-класс хранит

первый созданный тип. Он называется совместно используемым экземпляром. Однако вы можете запросить новый экземпляр. Это приведет к повторному исполнению функции для создания нового экземпляра, который не является совместно используемым.

Вот тестирование того, как происходит отображение. Базовый тип — это обычный тип `System.Object`, а отображенный тип является динамическим:

```
_target.Register<object>(tinyIoc => new {id = Guid.NewGuid()});
dynamic expected = _target.Resolve<object>();
Assert.IsInstanceOfType(expected.id, typeof (Guid),
  "Test failed: dynamic type was not returned.");
```

При *подготовке* к тестированию вспомогательный IoC-класс настраивается на возвращение динамического типа при запросе типа `object`. При *тестировании* выполняется вызов контейнера для получения экземпляра. И наконец, при *проверке* производится сравнение динамического типа со свойством `id` типа `Guid`, которое было реально возвращено.

Вы можете просмотреть оставшиеся тесты, чтобы изучить последовательность вызовов, возвращающих одни и те же (совместно используемые) экземпляры, в то время как запрос нового экземпляра выполняется так, как ожидается. Это довольно простые примеры тестов полностью самодостаточного компонента. А что происходит, если у вас есть компоненты наподобие `BlogDataSource`, которым для работы нужны другие компоненты? Здесь очень кстати будет вспомнить о пустышках и заглушках.

Пустышки и заглушки

При модульном тестировании вы сосредоточены на каком-то одном фрагменте кода. Главная мысль здесь заключается в том, чтобы протестировать основное назначение этого кода и не думать о внешних компонентах, которые могут затрагивать другие части приложения. В случае с классом `BlogDataSource` приходится задумываться о тех механизмах, которые он использует для чтения блогов, проверки кэша и составления списка групп и позиций, предоставляемых приложению. Класс не отвечает за то, как работает хранилище данных, он не осведомлен о том, как создаются диалоговые окна на целевой платформе. Решение этих задач возложено на другие компоненты.

Много споров вокруг точного определения того, что собой представляют пустышки и заглушки, но я склоняюсь к наиболее популярной трактовке: *заглушка* (stub) — это «фальшивая» реализация класса, которая служит исключительно для тестирования и благодаря которой зависящий от нее

класс может работать. Что касается *пустышки* (mock), то она являет собой нечто большее. Хотя это тоже фальшивка (то есть она создана исключительно для тестирования и не является частью системы), в дополнение к тому, что она позволяет функционировать другому классу, пустышка может отслеживать некоторые состояния, поэтому ее можно проверить на завершающем этапе теста (на этапе *проверки*). Это позволяет выяснить, правильно ли вызываются те или иные методы, и убедиться в правильной работе целевого компонента.

Существует несколько способов создания заглушек и пустышек. Первый заключается в написании собственных вспомогательных классов. Их вы можете найти в папке TestHelpers проекта WintellogTest. Класс `DialogTest` — это пример пустышки, так как он не только эмулирует асинхронное взаимодействие, но и осуществляет запись сообщений, которые можно исследовать в дальнейшем. Полностью код этого класса показан в листинге 9.1.

Листинг 9.1. Пустышка для имитации интерфейса IDialog

```
public class DialogTest : IDialog
{
  public string Message { get; set; }

  public Task ShowDialogAsync(string dialog)
  {
    return Task.Run(() =>
    {
      Message = dialog;
    });
  }
}
```

В пустышке не предусмотрен какой-либо пользовательский интерфейс, поэтому ее легко можно запустить при тестировании. Она сохранит переданное сообщение, поэтому вы можете проверить, верно ли зависимые компоненты взаимодействуют с интерфейсом. Другие классы предоставляют похожую функциональность, либо реализуя «фальшивые» источники данных для зависимых классов, либо записывая свойства, которые можно проверить позже.

Вы можете познакомиться с использованием заглушек в классе `BlogDataSourceTest`. Это один большой тест:

> *Дано*: есть новые позиции; *когда* заполняется группа блога, *тогда* общее количество позиций и количество новых позиций должно верно отражаться в итоговых показателях.

Тест имитирует существующие позиции в кэше и получение новой позиции из Интернета. Также он иллюстрирует недостаток текущего дизайна класса `BlogDataSource`. Класс пытается использовать `HttpClient` для получения страницы из тестовых данных, что невозможно, так как этого URL-адреса не существует. Хотя компонент хорошо обрабатывает такую ситуацию и в ходе теста выводится диалоговое окно с сообщением об ошибке, от подсистемы обращения к веб-ресурсам следовало бы абстрагироваться, скрыв ее за интерфейсом. То есть нужно создать пустышку для тестирования таких вызовов, контролировать корректность их выполнения и правильность возвращаемых ими данных.

На первом шаге определяется пример кэшированной позиции, то есть блог, и создается пустышка для хранилища данных, предоставляющая тестовый хэш-код:

```
var cached = new BlogItem {Id = Guid.NewGuid().ToString()};
var blog = new BlogGroup { Id = Guid.NewGuid().ToString() };
_storageUtilityTest.Items = new[] {"123"};
```

Затем пустышка для хранилища передается в функцию, которая вернет либо блог, либо кэшированную позицию в зависимости от того, что у нее запрашивается:

```
_storageUtilityTest.Restore = (type, folder, hashCode) =>
{
  if (type == typeof (BlogGroup))
  {
    return blog;
  }
  return cached;
};
```

И наконец, сымитированная новая позиция вместе с блогом добавляется в пустышку ненастоящего «нового элемента» и блога для того, чтобы предоставить их вспомогательному классу трансляции, который обрабатывает веб-каналы:

```
var newItem = new BlogItem {Id = Guid.NewGuid().ToString()};
_syndicationHelperTest.BlogItems.Add(newItem);
_syndicationHelperTest.BlogGroups.Add(blog);
```

Теперь мы начинаем с загрузки групп. Вызывается соответствующий метод и проводится проверка списка, позволяющая убедиться, что было обработано верное количество блогов:

```
await _target.LoadGroups();
Assert.AreEqual(1, _target.GroupList.Count,
  "Task failed: should have generated one group.");
```

Далее осуществляется обход групп для загрузки отдельных постов. Обратите внимание, что на этом шаге можно в соответствии с параметрами теста выполнить единственный вызов для одной группы, но я предпочитаю сделать его именно таким, чтобы добиться более точного соответствия с его реализацией в целевом приложении:

```
foreach (var group in _target.GroupList)
{
  await _target.LoadAllItems(group);
}
```

К данному моменту тест *подготовлен* и *выполнен*, поэтому пришло время для *проверки* результатов. Здесь проверяется количество позиций, производится сравнение списков и, наконец, запрашивается вывод диалогового окна, чтобы увидеть, будет ли оно вызываться, когда при попытке загрузки пустого URL-адреса произойдет ошибка. Одна только работа над этим учебным примером доказывает ценность тестирования. Оказалось, что в предыдущей версии приложения была ошибка, и сведения об общем количестве позиций отображались неверно. Тест позволил быстро выявить этот недочет, и я смог внести в программу исправления. В листинге 9.2 показаны итоговые проверки.

Листинг 9.2. Проверки для теста BlogDataSource

```
Assert.AreEqual(2, _target.GroupList[0].ItemCount,
  "Test failed: item count should have been 2.");
Assert.AreEqual(1, _target.GroupList[0].NewItemCount,
  "Test failed: new item count should have been 1.");
CollectionAssert.AreEquivalent(new[] {cached, newItem},
  _target.GroupList[0].Items,
  "Test failed: lists do not match.");
Assert.IsTrue(!string.IsNullOrEmpty(_dialogTest.Message),
  "Test failed: dialog was not called with error message.");
```

Существует множество платформ, упрощающих создание пустышек и заглушек. Microsoft Fakes и MOQ — это лишь пара примеров таких платформ. Они позволяют динамически сконфигурировать пустышку или заглушку для использования в модульных тестах. Это избавит вас от необходимости создавать сложные пустышки или заглушки самостоятельно, и вы сможете настраивать режимы их работы так, чтобы они соответствовали целям

теста, который вы пишете. В примерах этой главы был задействован проект модульного тестирования приложений для Магазина Windows. Если для целей совместного использования кода вы применяете переносимую библиотеку классов, можете столь же просто создать проект для модульного тестирования традиционных настольных приложений и строить тесты на его основе, а также выбирать платформы для разработки тестов и пустышек.

Выводы

В этой главе вы узнали о паттерне MVVM и о том, какие преимущества вы можете получить, разрабатывая приложения с его использованием. Я показал, как с помощью переносимой библиотеки классов (PCL) писать программный код, пригодный для совместного использования на различных платформах, как создать WPF-приложение, применяя те же основные классы, что и в приложении для Windows 8. Также вы узнали о выгодах, которые может принести тестирование приложений, и о нескольких методах создания модульных тестов в Visual Studio 2012. Кроме того, вы познакомились с пустышками, которые при тестировании классов с зависимостями позволяют избавиться от зависимостей.

В следующей главе вы узнаете, как подготовить ваше приложение для Магазина Windows. Мы рассмотрим особенности установки цены приложения, поговорим о покупках из приложения, с помощью которых пользователь активирует какие-либо недоступные ранее инструменты приложения (я называю их «продуктами»). Поговорим мы и о тестировании этой функциональности. Я продемонстрирую использование комплекта сертификации приложений для Windows (Windows App Cert Kit), с помощью которого можно проверить приложение перед передачей в Магазин Windows. Вы узнаете и о том, как, минуя Магазин Windows, создать распространяемый пакет для совместного использования кода.

Пакетирование и развертывание 10

Когда ваше приложение готово, вам нужно найти способ доставить его потенциальным клиентам. Это можно сделать разными способами, включая так называемую параллельную загрузку (side-loading), о которой мы поговорим в этой главе позже. Многим разработчикам захочется вступить в партнерские отношения с Microsoft, это можно сделать посредством Магазина Windows. Доступ в Магазин можно получить как через набор онлайновых страниц, который создается для вашего приложения, так и посредством предустановленного приложения для Windows 8 под названием Store (Магазин).

Идея Магазина приложений витала в воздухе уже несколько лет, став популярной благодаря различным мобильным платформам. Магазин — это единый сервис для управления приложениями, их покупки, загрузки и развертывания. Такая концепция предоставляет пользователям единое место для поиска приложений и обеспечивает высокий уровень их безопасности, так как приложения, перед тем как они становятся доступными для загрузки, тщательно проверяют и тестируют. Больше о проверке приложений, отправленных в Магазин Windows, вы узнаете далее в данной главе.

Магазин Windows

Магазин Windows предлагает отдельным разработчикам и организациям сервисы для публикации их приложений (сюда входят как приложения для Windows 8, так и традиционные настольные приложения). Пользователям

он позволяет искать приложения по каталогу, покупать и устанавливать их. Магазин поддерживает инфраструктуру поиска приложений в онлайновом режиме и с помощью приложения для Windows 8. Его возможности весьма широки, и для разработчиков, которые решат использовать его для продажи приложений, это означает потенциально высокую прибыль. Чтобы получать прибыль от своего приложения для Windows 8, можно следовать различным бизнес-моделям, например встраивать в приложение рекламные объявления, пользуясь либо собственной рекламной сетью Microsoft, либо сторонними сервисами, такими как AdDuplex (www.adduplex.com).

Поиск приложений

Важно, чтобы потенциальный покупатель мог не только легко найти ваше приложение, но так же легко приобрести и установить его. Windows 8 и Internet Explorer 10 работают вместе, предоставляя все необходимые инструменты для плавного перехода от работы в онлайновом режиме к локальному приложению Магазина Windows. Когда вы отправляете приложение в Магазин Windows, оно становится доступным и в онлайновом режиме, и посредством соответствующего приложения Магазина Windows.

Для того чтобы увидеть, как работает подобная интеграция, откройте Internet Explorer 10 с начального экрана (то есть запустите его версию для Windows 8, а не настольную версию) и перейдите к странице приложения Cut the Rope (http://cuttherope.ie).

Откройте с помощью жеста скольжения панель приложения, и вы увидите, что у кнопки с гаечным ключом появился знак «плюс» (+), указывающий на наличие дополнительных возможностей. Когда вы коснетесь гаечного ключа, то увидите несколько пунктов меню, в том числе тот, который позволяет установить на компьютер приложение для этого веб-сайта, как показано на рис. 10.1.

Выбрав команду Get app for this site (Получить приложение для этого сайта), вы перейдете на соответствующую страницу Магазина Windows, где легко сможете загрузить приложение. Когда приложение будет установлено, меню расширенных возможностей в Internet Explorer даст вам возможность запустить с его помощью локальную версию приложения. Хотите узнать, как сайты поддерживают подобную интеграцию? Используйте команду View on the desktop (Просмотреть на рабочем столе), которая имеется в меню кнопки с гаечным ключом, и откройте код страницы, щелкнув в любом месте страницы правой кнопкой мыши и выбрав команду View Source (Просмотр HTML-кода). В HTML-коде вы увидите несколько мета-тегов, которые показаны в листинге 10.1.

Рис. 10.1. Пример интеграции веб-сайта и Магазина Windows

Листинг 10.1. HTML-код для интеграции веб-сайта Cut the Rope и Магазина Windows

```
<meta name="application-name" content="Cut the Rope" />
<meta name="msapplication-tooltip" content="Play Cut the Rope! A
   ↳ mysterious package has arrived, and the little monster inside
   ↳ Has only one request… CANDY!" />
<meta name="msapplication-navbutton-color" content="#659729" />
<meta name="msApplication-ID" content="App"/>
<meta name="msApplication-PackageFamilyName"
  content="ZeptoLabUKLimited.CutTheRope_sq9zxnwrk84pj"/>
<link rel="shortcut icon" href="/favicon.ico"
    type="image/x-icon" />
```

Теги обеспечивают контекст, который Internet Explorer 10 может задействовать, чтобы предоставить конечному пользователю необходимую ссылку. Ключевой тег — `msApplication-PackageFamilyName`, его можно найти в манифесте приложения. Благодаря имени приложения из этого тега, Internet Explorer может определить, установлено ли приложение, и, если

нет, предоставить его идентификатор для Магазина Windows, чтобы вы могли его установить.

Для того чтобы увидеть еще один пример интеграции, откройте сервис Bing из версии Internet Explorer 10 для Windows 8 и выполните поиск по ключевому слову `Rowi`. Так называется мое любимое приложение для работы с сервисом Twitter. Прокрутите страницу. Среди результатов поиска имеется значок приложения из Магазина Windows, сводка оценок и ссылка для загрузки (рис. 10.2).

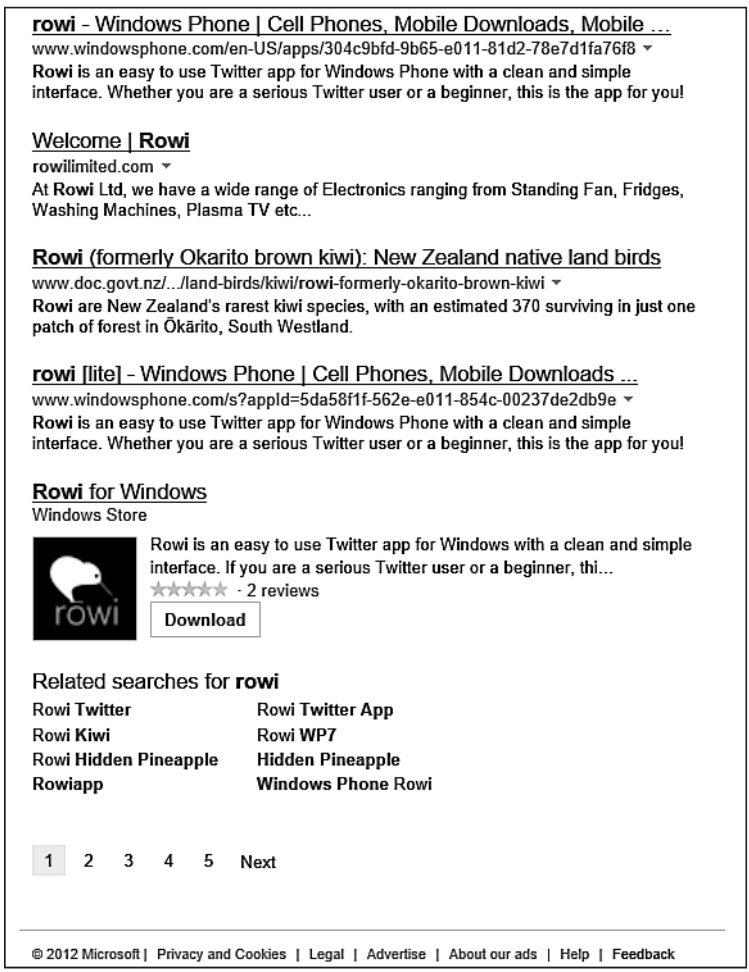


Рис. 10.2. Результаты Bing-поиска приложения для Windows 8 под названием Rowi, предназначенного для работы с сервисом Twitter (Используется с разрешения Hidden Pineapple, LLC)

Если вы щелкнете на ссылке, откроется страница данного приложения в программе Store (Магазин), которая установлена у вас на компьютере. Так происходит потому, что URL-адрес начинается с префикса `ms-windows-store` и предоставляет уникальный идентификатор, который можно передать приложению Магазина Windows, чтобы оно открыло страницу приложения (рис. 10.3). Это упрощает просмотр сведений о приложении. Для того чтобы его установить, достаточно щелкнуть на кнопке Install (Установить).

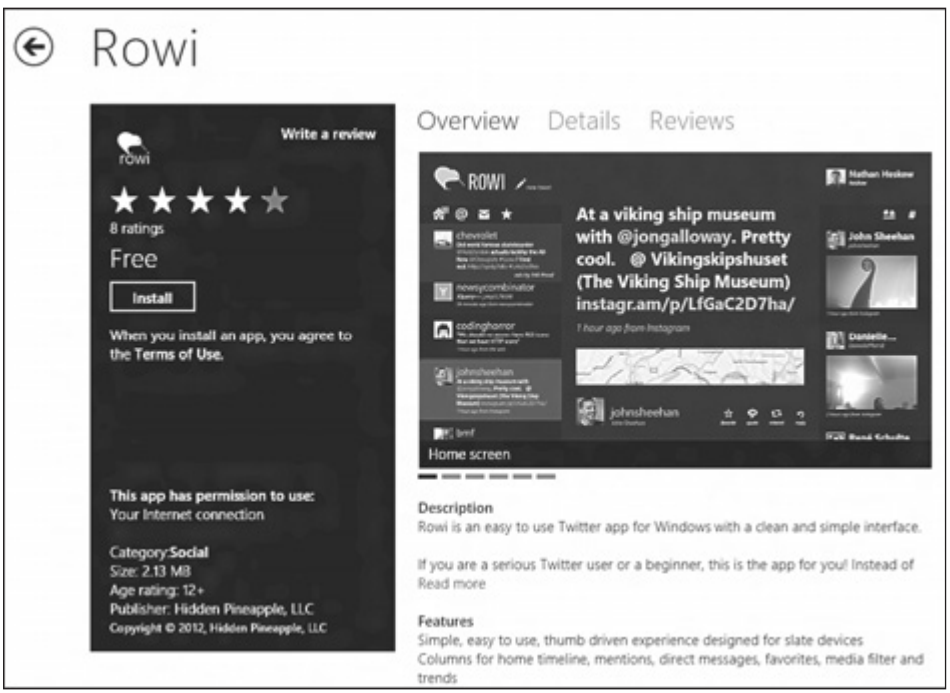


Рис. 10.3. Страница приложения Rowi в Магазине Windows (Используется с разрешения Hidden Pineapple, LLC)

Конечно, легче всего найти приложение непосредственно в самом Магазине Windows. Главная страница содержит подборку популярных приложений и позволяет просматривать различные категории, от приложений, набравших самые высокие оценки, до недавно добавленных. Магазин Windows, кроме того, поддерживает семантическое масштабирование, предлагая пользователю быстрый доступ к различным категориям программ, как показано на рис. 10.4.

Далее в этой главе вы узнаете, как включить приложение в список Магазина Windows. Но для начала можете спросить себя: «А нужен ли мне список Магазина Windows?» Для того чтобы ответить на этот вопрос, в первую

очередь неплохо бы узнать об охвате Магазином Windows рынков разных стран и о стоимости подписки на его сервисы для разработчика.

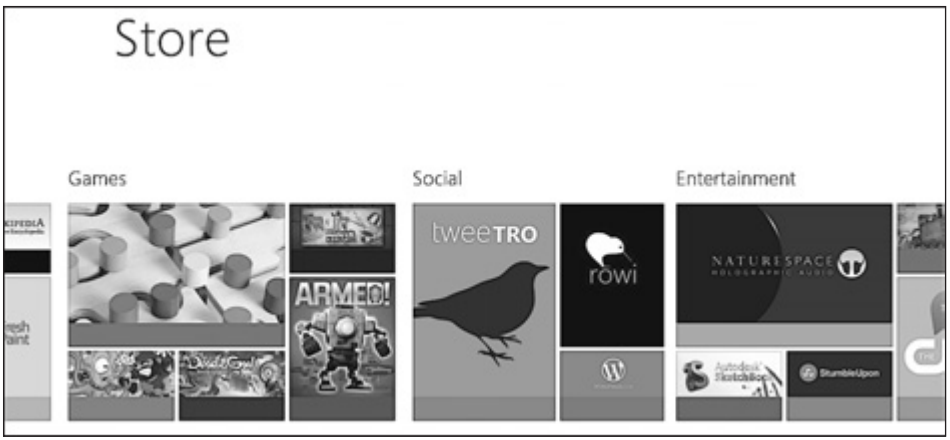


Рис. 10.4. Семантическое масштабирование в Магазине Windows

Охват

По данным Microsoft, Магазин Windows позволяет разработчикам продавать их приложения более чем на 200 рынках, поддерживая около 70 национальных валют и возможность локализации приложений примерно для 100 языков. Это довольно серьезный охват рынка, подробности вы можете найти на странице http://blogs.msdn.com/b/windowsstore_ru/archive/2012/01/10/10255231.aspx.

По данным Apple, в марте 2012 года было продано 365 миллионов устройств под управлением iOS (http://techcrunch.com/2012/06/11/apple-wwdc2012-iphones-ipads-sold/). Сравните это с более чем 600 миллионами лицензий на Windows 7, проданных в июне 2012 года (http://www.engadget.com/2012/06/06/over-600-million-windows-7-licenses-sold/). Это огромное количество устройств. Windows 8 (за исключением платформы Windows RT, ориентированной на ARM-процессоры) поддерживает обратную совместимость с устройствами, на которых работает Windows 7. Таким образом, для разработчика это потенциально огромный рынок. Если бы некое приложение для Windows 7 получило всего 0,1 % этого рынка, то количество его копий составило бы 600 000.

В Магазине Windows вы можете назначать цену за лицензию на программный продукт в диапазоне от 1,49 до 999,99 $, то же касается стоимости покупок из приложения, которые обычно используются для активизации

дополнительных инструментов. В следующем разделе этой главы мы обсудим данные вопросы подробнее. Здесь применяется такая модель разделения доходов, в которой на начальном этапе Microsoft получает 30 %, а разработчик — 70 %. Когда объем продаж разработчика достигает $25 000, Microsoft получает 20 %, а разработчик — оставшиеся 80 %. Если в предыдущем примере представить, что вы назначили самую низкую цену за программу ($1,49 за лицензию), то 600 000 установленных копий *после* разделения доходов с Microsoft принесут вам $700 000.

Для того чтобы зарабатывать на приложениях, вы должны оплатить учетную запись разработчика Microsoft. Это позволит вам отправлять приложения в Магазин Windows. Сейчас существует две ценовые модели: $49 для отдельных разработчиков и $99 для организаций. Та и другая позволяют разместить в Магазине Windows несколько приложений.

Бизнес-модели

Существует множество бизнес-моделей, из которых вы можете выбрать подходящую именно для вашего приложения. Сначала нужно определить общую стоимость приложения. Задав стоимость приложения, вы можете указать, предусмотрен ли для него пробный период. Пробный период может означать ограничения либо по времени, либо по возможностям. В первом случае пользователь получает полнофункциональное приложение, но для того чтобы работать с ним по прошествии некоторого времени, он должен совершить покупку. Во втором случае некоторые инструменты программы будут недоступны до тех пор, пока пользователь ее не купит. Пример задания этих параметров представлен на рис. 10.5.

Прикладной программный интерфейс для создания пробной версии приложения и перехода к полной версии устроен весьма просто. Сначала нужно перейти по ссылке http://code.msdn.microsoft.com/windowsapps/Licensing-API-Sample-19712f1a, чтобы загрузить один из SDK-примеров, который демонстрирует работу пробной версии приложения и организацию покупок из приложения для активации дополнительных инструментов.

В примере показано, как пользователь, работая с пробной версией приложения, может нажать кнопку для покупки полной версии продукта. Первый шаг здесь заключается в получении информации о лицензии приложения:

```
var licenseInformation = CurrentApp.LicenseInformation;
```

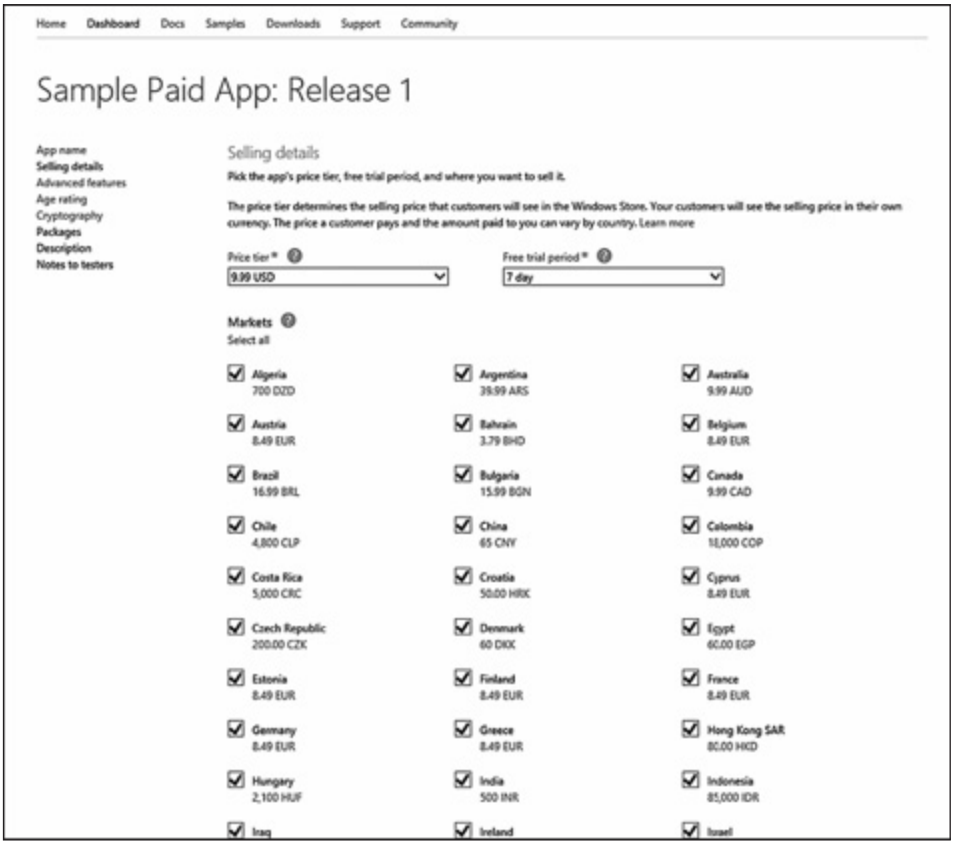


Рис. 10.5. Задание первоначальной стоимости приложения и параметров его пробной версии

Если приложение работает в пробном режиме, производится запрос на покупку полной версии. Все остальное выполняется с помощью внутренних механизмов Магазина Windows:

```
if (licenseInformation.IsTrial)
{
  await CurrentApp.RequestAppPurchaseAsync(false);
}
```

Ваше приложение не сможет взаимодействовать с сервером лицензирования до тех пор, пока оно не окажется в Магазине Windows. Для того чтобы протестировать нечто вроде перехода от пробной версии продукта к полной или покупку из приложения, можете воспользоваться встроенным имитатором. Для этого следует заменить все ссылки на `CurrentApp` ссылками на `CurrentAppSimulator` (их вы можете увидеть в загруженном примере).

Имитатор позволяет загрузить специальный XML-документ, предназначенный для задания состояния лицензии и параметров покупки приложения. В листинге 10.2 такой файл показан для приложения, которое поддерживает пробный режим работы, при этом для того, чтобы воспользоваться полной версией приложения, его нужно купить за $4,99.

Листинг 10.2. Пример XML-документа, который позволяет указывать параметры лицензирования, применяемые при тестировании

```
<?xml version="1.0" encoding="utf-16" ?>
<CurrentApp>
  <ListingInformation>
    <App>
      <AppId>2B14D306-D8F8-4066-A45B-0FB3464C67F2</AppId>
      <LinkUri>http://apps.microsoft.com/app/
          ↳ 2B14D306-D8F8-4066-A45B-0FB3464C67F2</LinkUri>
      <CurrentMarket>en-US</CurrentMarket>
      <AgeRating>3</AgeRating>
      <MarketData xml:lang="en-us">
        <Name>Trial management full license</Name>
        <Description>Sample app for demonstrating trial
            ↳ license management</Description>
        <Price>4.99</Price>
        <CurrencySymbol>$</CurrencySymbol>
      </MarketData>
    </App>
  </ListingInformation>
  <LicenseInformation>
    <App>
      <IsActive>true</IsActive>
      <IsTrial>true</IsTrial>
      <ExpirationDate>2013-01-01T00:00:00.00Z</ExpirationDate>
    </App>
  </LicenseInformation>
</CurrentApp>
```

Данный файл загружается в имитатор приложения с помощью кода, размещенного в файле TrialMode.xaml.cs:

```
StorageFolder proxyDataFolder = await
  Package.Current.InstalledLocation.GetFolderAsync("data");
StorageFile proxyFile = await proxyDataFolder
  .GetFileAsync("trial-mode.xml");
await CurrentAppSimulator.ReloadSimulatorAsync(proxyFile);
```

Для того чтобы протестировать разные сценарии, можете перезагружать данные имитатора как угодно часто.

СОВЕТ

Имитатор находится в оперативной памяти. Когда вы загружаете файл с данными о лицензии, вы можете производить изменения, например для имитации покупок из приложения. Эти изменения сохраняются в оперативной памяти, не оказывая влияния на файл. Каждый раз, когда вы перезапускаете приложение, сведения о лицензии и покупках возвращаются к исходному состоянию, заданному в XML-файле. Если вам нужно протестировать различные сценарии, можете использовать несколько файлов, как показано в SDK-примере. В примере загружается подходящий для тестируемого сценария файл, а при необходимости, чтобы вернуть все в исходное состояние, этот файл перезагружается. Перед отправкой приложения в Магазин Windows не забудьте удалить тестовые файлы и переименовать все ссылки на имитатор приложения.

В коде также задан обработчик события, который реагирует на каждое изменение лицензионной информации, например при покупке в режиме имитации (или при реальной покупке, когда приложение выйдет в свет):

```
licenseChangeHandler = new
  LicenseChangedEventHandler(TrialModeRefreshScenario);
CurrentAppSimulator.LicenseInformation.LicenseChanged +=
  licenseChangeHandler;
```

Имитатор позволяет указать результат тестовой покупки. Например, вы можете сообщить ему о том, что покупка не состоялась, чтобы проверить, верно ли приложение обрабатывает такую ситуацию. Имитатор показан на рис. 10.6. Когда нажимают кнопку, чтобы совершить покупку, появляется диалоговое окно Магазина Windows с раскрывающимся списком, позволяющим выбрать желаемый результат покупки.

Пробный срок играет весьма важную роль. В соответствии со статистикой Microsoft на примере платформы Windows Phone, приложения, которые имеют пробный срок использования, загружают в 70 раз чаще, чем другие. При этом 10 % загрузок пробных версий трансформируются в покупку, что позволяет таким приложениям приносить в 10 раз больший доход, чем приносят продукты, не имеющие пробного срока. Подробности вы можете узнать на странице http://bit.ly/ekk9s5.

Ваше приложение может предлагать пользователю конкретные «продукты» или функциональные возможности, которые можно купить. Это позволит пользователю настраивать приложение по своему вкусу и покупать

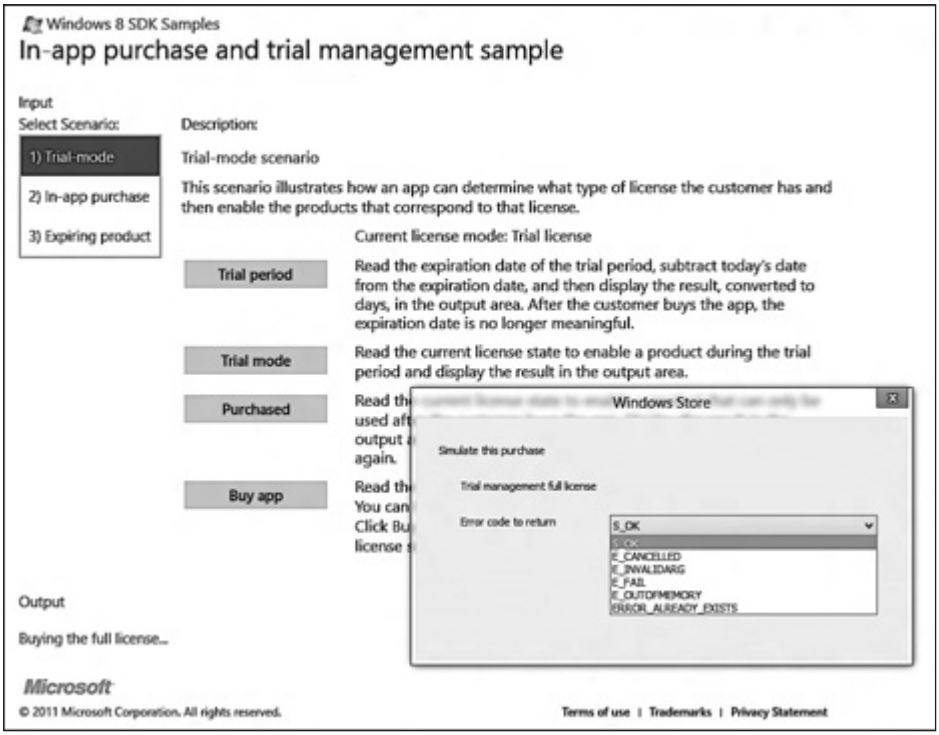


Рис. 10.6. Имитатор Магазина Windows

только те инструменты программы, которые ему нужны. Например, редактор изображений может по умолчанию предоставлять средства для работы с некоторым набором графических форматов. В то же время, чтобы у пользователя была возможность работать с форматами, которые, вероятнее всего, пригодятся профессиональным дизайнерам, а не «домашним умельцам», ему нужно совершить соответствующую покупку. Настройка цифровых продуктов осуществляется средствами Магазина Windows, которые становятся доступными при входе от имени учетной записи разработчика. Однако того же эффекта в тестовых целях можно добиться с помощью простого XML-файла:

```
<Product ProductId="product1">
  <MarketData xml:lang="en-us">
    <Name>Product 1</Name>
    <Price>1.99</Price>
    <CurrencySymbol>$</CurrencySymbol>
  </MarketData>
</Product>
```

В листинге 10.3 показано, как запросить лицензионную информацию о продукте для ее показа пользователю.

Листинг 10.3. Запрос информации о продукте

```
ListingInformation listing = await
  CurrentAppSimulator.LoadListingInformationAsync();
var product1 = listing.ProductListings["product1"];
var product2 = listing.ProductListings["product2"];
Product1SellMessage.Text = "You can buy " + product1.Name +
  " for: " + product1.FormattedPrice + ".";
Product2SellMessage.Text = "You can buy " + product2.Name +
  " for: " + product2.FormattedPrice + ".";
```

Продукт предоставляет свойство `IsActive`, с помощью которого можно узнать, был ли он куплен (активирован). Вы можете запросить это свойство и отключить соответствующую функциональность до момента, когда пользователь совершит покупку. Запрос на покупку продукта выполняется так же, как и на покупку приложения. Единственная разница заключается в параметре, который указывает на конкретный продукт:

```
await CurrentAppSimulator.RequestProductPurchaseAsync(
  "product1", false);
```

Кроме того, можно задать срок действия продуктов. Это позволит получать регулярный доход, предоставляя клиентам продукты, действующие определенное время. После того как данный срок истечет, пользователю придется повторить покупку, чтобы получить доступ к функциональности, которую она активирует. У продуктов есть свойство `ExpirationDate`, которое позволяет сообщить пользователю о дате истечения срока действия продукта и о том, требуется ли повторная оплата.

Реклама в приложениях

Реклама — один из инструментов монетизации приложений. Такой подход вполне обычен — приложение можно поддерживать за счет отчислений за показ в нем рекламы. Кроме того, можно предоставлять бесплатную версию, содержащую рекламу, а после покупки платной версии отключать рекламу. Microsoft предлагает два варианта показа рекламы в приложениях для Windows 8. Первый заключается в использовании рекламных сервисов сторонних производителей, второй — собственного сервиса Microsoft, который доступен на странице www.windowsadvertising.com.

Для того чтобы приступить к работе с рекламной платформой Microsoft, сначала нужно перейти по этой ссылке и загрузить SDK. В результате у вас будут инструменты, необходимые для интеграции рекламной платформы в ваше приложение. Когда у вас будет SDK, изучите документацию на странице http://msdn.microsoft.com/en-us/library/hh506371(MSADS.10).aspx.

Полный обзор средств использования рекламы в приложениях, написанных на XAML и C#, можно найти на странице http://msdn.microsoft.com/en-us/library/hh506359(v=msads.10).aspx.

Вам также нужно создать учетную запись. Это позволит, во-первых, указать, как вы собираетесь получать отчисления за показ рекламы в приложениях. Во-вторых, вы получите возможность отслеживать эффективность рекламы в приложениях. Если вы решите воспользоваться каким-то другим рекламным сервисом, таким как AdDuplex, вам, скорее всего, понадобится загрузить его SDK и следовать инструкциям по интеграции подсистемы рекламы с вашим приложением. Магазин Windows достаточно гибок, чтобы в нем можно было размещать приложения с рекламными объявлениями от любого сервиса, но эти объявления должны соответствовать правилам размещения приложений в Магазине, о которых вы узнаете далее. Поставщику рекламных объявлений можно предоставить сведения о приложении, что поможет ему соответствующим образом настраивать вывод объявлений, а это, весьма вероятно, приведет к увеличению количества «кликов» и, как результат, — к увеличению вашего дохода.

Подготовка приложения для Магазина Windows

Для того чтобы подготовить приложение к отправке в Магазин Windows, нужно предпринять несколько шагов. Для начала ознакомьтесь с правилами сертификации приложений, которые можно найти на странице http://msdn.microsoft.com/ru-ru/library/windows/apps/hh694083.aspx.

В соответствии с правилами есть несколько ключевых требований, которым должно соответствовать приложение для того, чтобы его включили в каталог Магазина Windows. Шесть из них, которые применимы к приложениям для Windows 8, я кратко описываю далее.

Обеспечьте ценность приложения

Приложение должно предоставлять конечному пользователю полную функциональность. В нем не должно быть мест, находящихся «в процессе разработки», или мест, которые выглядят незавершенными. Если вы предлагаете пробный режим, его возможности должны напоминать

возможности полнофункциональной версии приложения. При этом можно либо предоставить полную функциональность на ограниченное время, либо ограничить доступ к некоторым инструментам приложения.

Предлагайте нечто большее, нежели показ рекламы и вывод веб-страниц

Приложение может содержать рекламные объявления, но должно предоставлять и полезные инструменты. Приложения, которые не делают ничего полезного, исключая показ рекламы, не принимаются для размещения в Магазине. Кроме того, для показа пользователю дополнительной рекламы нельзя применять плитки, заголовок приложения, описание, уведомления и панель приложения. Нельзя разрабатывать приложения, которые, не делая ничего полезного, занимаются исключительно тем, что загружают и выводят на экран веб-страницы.

Нужно быть предсказуемым

Приложение должно создаваться с учетом рекомендаций по разработке для Windows 8, о которых мы говорили в этой книге. В частности, оно должно поддерживать работу при различной ориентации экрана и в различных режимах просмотра (включая режим фиксации). Приложение должно верно обрабатывать события завершения и приостановки. При этом оно должно обрабатывать ошибки, такие как потеря сетевого соединения, не зависая и не прекращая реагировать на действия пользователя. Обновления должны расширять возможности приложения, делать его более стабильным и никогда не должны вводить какие-либо ограничения на его функциональность. Важно, чтобы вы использовали в приложении для Windows 8 только прикладные программные интерфейсы среды WinRT.

Держите пользователя в курсе дела

Приложение должно следовать рекомендациям по уведомлению пользователя о применении своих инструментов, таких, например, как API для определения местоположения, и давать пользователю возможность отключать уведомления. Если приложение собирает персональную информацию, вы должны предложить политику конфиденциальности и соответствовать законодательству в этой области. Если вы собираетесь публиковать персональную информацию пользователя, дайте ему самому решить, нужно ли это делать. Приложение не должно содержать вредоносного кода и предлагать пользователю совершать действия, которые могут повредить его устройства.

Ориентируйтесь на глобальную аудиторию

Приложение не должно содержать материалы для взрослых. Метаданные приложения должны ориентироваться на пользователей всех возрастных групп. Наивысший рейтинг, который может получить контент приложения или его метаданные в Магазине Windows, составляет 12+. То есть они должны содержать информацию, которая подходит для пользователей от 12 лет и старше. Приложение не должно содержать материалы, пропагандирующие дискриминацию, ненависть или насилие. Программный продукт не должен поощрять нелегальную деятельность или содержать непристойные материалы. Полный список ограничений вы можете прочитать, перейдя по приведенной ранее ссылке.

Нужно быть понятным и легко узнаваемым

У приложения должны быть легко узнаваемая уникальная плитка и описание. Вы должны предоставить техническую поддержку. Кроме того, при публикации приложения вы должны указать как минимум один рынок, на котором оно будет распространяться, и локализовать его для всех поддерживаемых им языков. Более того, когда вы предоставляете обновления, вы должны полностью описывать внесенные изменения. Также вы должны при отправке программы в магазин предоставить по меньшей мере одну копию его экрана. Как это сделать, вы узнаете в следующем разделе.

Передача приложения в Магазин Windows

Полностью последовательность действий для передачи приложения в Магазин Windows описана на странице http://msdn.microsoft.com/ru-ru/library/windows/apps/hh454036(v=vs.110).aspx/.

Для этого вам понадобится выполнить следующие шаги:

1. Зарегистрируйте учетную запись разработчика. Помните о том, что плата за регистрацию индивидуального разработчика составляет $49, за регистрацию организации — $99.
2. Зарезервируйте название для своего приложения. Резервировать имена можно максимум за год до отправки приложения в магазин. Подробности о названиях приложений и соответствующие руководства вы можете найти на странице http://msdn.microsoft.com/ru-ru/library/windows/apps/hh694077.aspx.
3. Получите лицензию разработчика.
4. Соответствующим образом отредактируйте манифест приложения.

5. Свяжите приложение с Магазином Windows. Для того чтобы это сделать, щелкните правой кнопкой мыши на проекте приложения в обозревателе решений и в появившемся меню выберите команду Store ▸ Associate App with the Store (Магазин ▸ Связать приложение с Магазином), как показано на рис. 10.7.

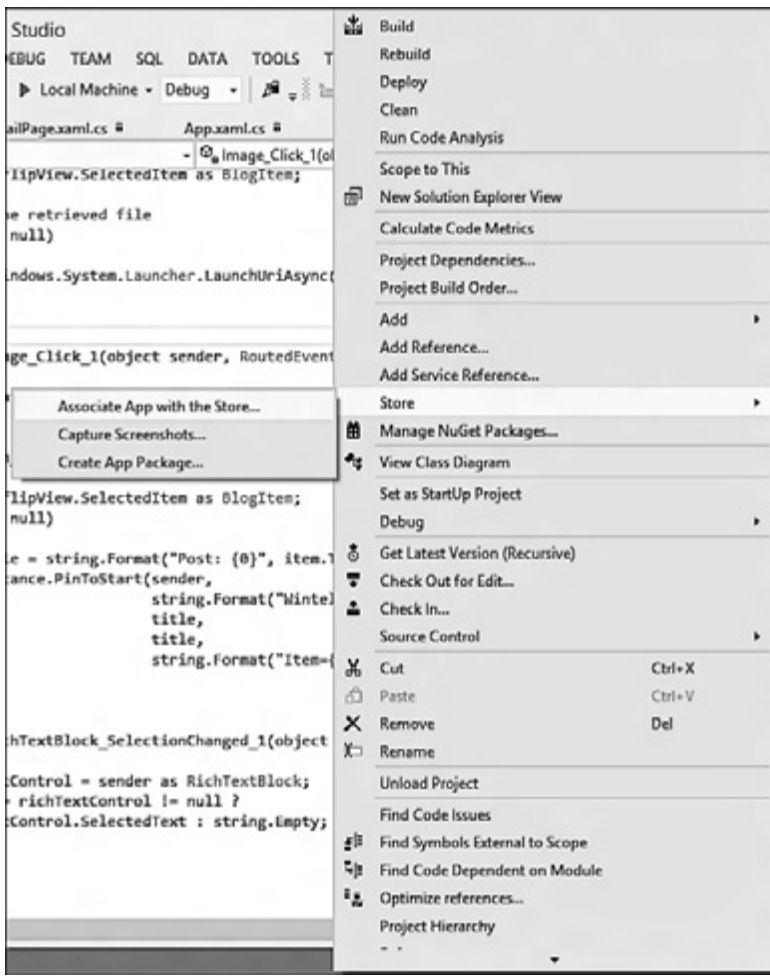


Рис. 10.7. Связывание приложения с Магазином Windows

6. Создайте снимки экрана приложения. Для этого щелкните правой кнопкой мыши на проекте приложения в обозревателе решений и в появившемся меню выберите команду Store ▸ Capture Screenshots (Магазин ▸ Сделать снимки экрана). Будет запущен имитатор, с помощью которого вы можете сделать копии экрана, нажимая кнопку с изображением камеры на панели инструментов, которая находится справа.

7. Создайте пакет приложения. Вызовите контекстное меню проекта и выберите в нем команду Store ▸ Create App Package (Магазин ▸ Создать пакеты приложения). Вам будет предложено авторизоваться с использованием учетной записи разработчика.
8. Отправьте пакет приложения в Магазин Windows с помощью учетной записи разработчика.

Комплект сертификации приложений для Windows

Для того чтобы помочь вам удостовериться в том, что ваш программный продукт готов к передаче в Магазин Windows, можете воспользоваться комплектом сертификации приложений для Windows (Windows App Certification Kit). Данный набор инструментов выполняет анализ приложения и определяет следующее:

- Верно ли настроен манифест приложения, в том числе описаны ли возможности и объявления.
- Все ли ресурсы приложения доступны и не содержат ли ошибок.
- Нормально ли запускается приложение и не зависает ли оно.
- Скомпилировано ли приложение в конфигурации выпуска (release), а не в конфигурации отладки (debug).
- Правильная ли кодировка в файлах приложения.
- Не занимают ли запуск и приостановка приложения больше времени, чем регламентировано системой.
- Вызывает ли приложение нужные прикладные программные интерфейсы среды WinRT, не пытается ли оно вызывать запрещенные интерфейсы или интерфейсы, не рассчитанные на приложения для Магазина Windows.
- Использует ли приложение необходимые для его работы компоненты системы безопасности Windows.

Для того чтобы начать тестирование, найдите и запустите программу «Windows App Cert Kit» или «appcertui.». После запуска выберите вариант Validate Windows Store App (Проверить приложение Магазина Windows), как показано на рис. 10.8 (можно также выполнять проверку приложений других типов, например настольные приложения).

Через некоторое время после выбора этого варианта откроется диалоговое окно со списком установленных в вашей системе приложений для Windows 8. Для тестирования вы можете выбрать одно или несколько. На рис. 10.9 показано, что для тестирования выбрано приложение Wintellog Mvvm, о котором рассказывается в главе 9.

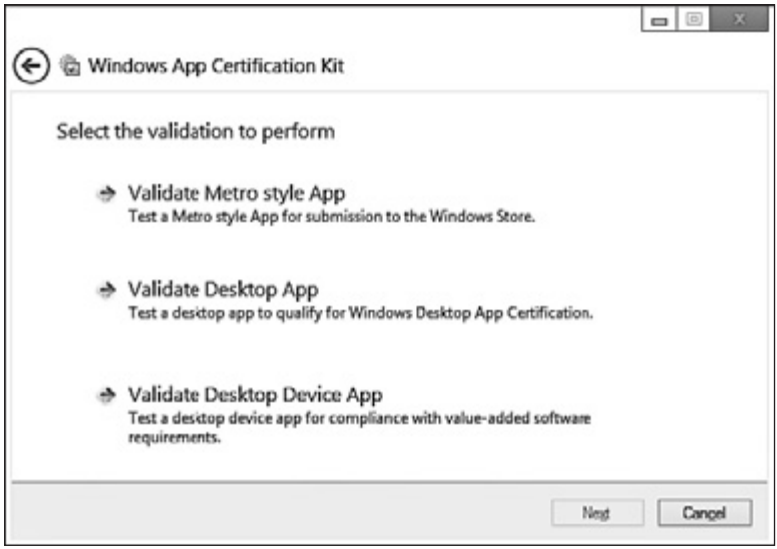


Рис. 10.8. Проверка приложения перед отправкой в Магазин Windows

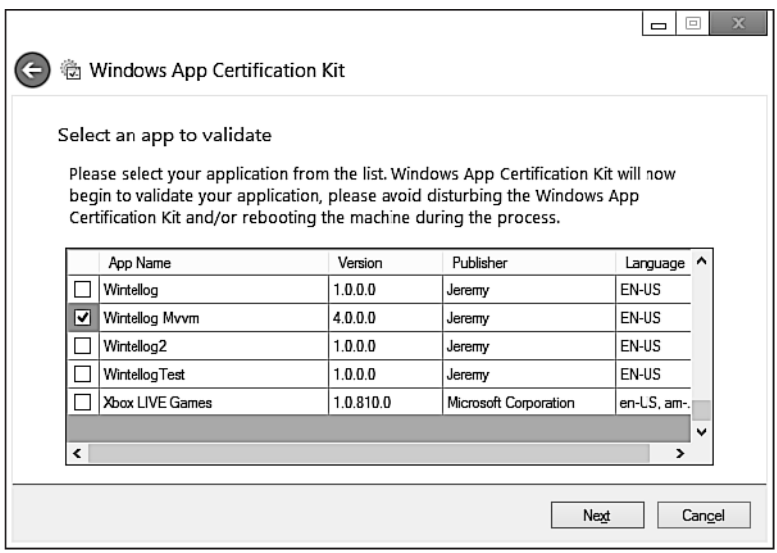


Рис. 10.9. Выбор приложения для тестирования

Щелкните на кнопке Next (Далее), чтобы начать тестирование. Оно может занять несколько минут в зависимости от того, какое приложение и на каком оборудовании вы тестируете. Важно, чтобы во время тестов вы не пытались работать с программой. В процессе тестирования приложение несколько раз будет перезапущено. Лучшее, что вы можете сделать, — просто позволить тестам выполняться, дожидаясь их окончания.

Когда тесты завершатся, вам будет предложено указать место для сохранения XML-отчета о тестировании. При этом будет также создан локальный HTML-файл с результатами. Вы можете щелкнуть на ссылке, которую увидите после завершения тестирования, чтобы просмотреть результаты. Если какой-либо из тестов пройден не будет, в описании результатов вы найдете рекомендации по исправлению ошибок. Вам следует добиться полного прохождения всех тестов перед отправкой приложения в Магазин Windows. На рис. 10.10 показаны результаты испытаний программы Wintellog Mvvm из главы 9.

Рис. 10.10. Результаты испытаний приложения с помощью комплекта сертификации приложений для Windows

Подробнее о комплекте сертификации приложений для Windows вы можете узнать на странице http://msdn.microsoft.com/ru-ru/library/windows/apps/hh694081.aspx.

В статье можно найти ссылки на страницы с подробным описанием выполняемых тестов. Там есть разъяснения по поводу того, что именно подвергается испытаниям, и руководства по решению проблем.

Чего ждать дальше

В центре разработчиков вы можете найти полный обзор процедуры отправки приложения, а также примерное время, которое могут занять те или иные шаги. Вы будет извещаться о каждом шаге этой процедуры. На рис. 10.11 показан пример информационной панели, иллюстрирующей продвижение приложения по этапам сертификации, включая этапы, которые ему еще предстоит пройти.

Рис. 10.11. Процедура отправки приложения

Когда приложение будет принято к публикации, появится дополнительная информационная панель, где вы можете найти сведения о покупках и установках приложения. Вы сможете отбирать данные по отдельным регионам, по возрастным группам пользователей и по другим демографическим показателям, чтобы лучше понять, как пользователи приняли ваше приложение. Также вы сможете увидеть оценки приложения и, конечно, наблюдать за доходами, которые оно приносит. На рис. 10.12 показан пример такой информационной панели.

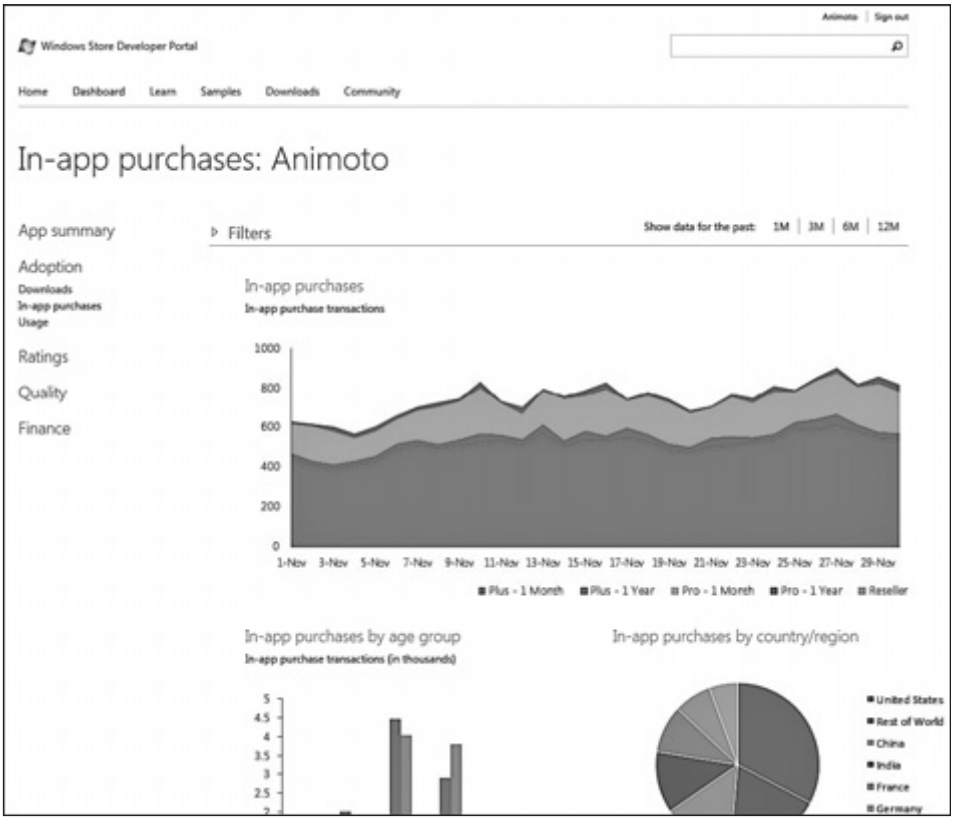


Рис. 10.12. Информационная панель Магазина Windows для опубликованного приложения

Иногда возникает необходимость распространять приложения, минуя Магазин Windows. Например, так может произойти, если у вас есть ранняя версия продукта, которую вы хотите передать кому-либо для тестирования или пробной работы. Так бывает и тогда, когда вы разрабатываете набор бизнес-приложений для локальной сети организации. К счастью, все это возможно благодаря так называемой параллельной загрузке (side-loading). При таком подходе к распространению приложений у вас появляется возможность создать их установочные пакеты, которые можно напрямую развертывать на других устройствах.

Параллельная загрузка

Вы можете устанавливать приложения с помощью инструмента Windows PowerShell. Хотя приложения, распространяемые таким образом, не прохо-

дят сертификацию Магазина Windows и не устанавливаются с его помощью, они должны быть подписаны доверенным сертификатом. И установить их можно только на компьютере, который доверяет этому сертификату.

Подробности об управлении сертификатами вы можете найти на странице http://msdn.microsoft.com/en-us/library/aa376553(v=vs.85).aspx.

Узнать о том, как подписывать приложения для Windows 8, можно на странице http://msdn.microsoft.com/en-us/library/aa387764(v=vs.85).aspx.

И наконец, руководства по организации параллельной загрузки приложений находятся на странице http://technet.microsoft.com/ru-ru/library/hh852635.aspx.

Первый шаг в подготовке к параллельной загрузке приложения заключается в создании пакета, который можно распространять. Это можно сделать, щелкнув правой кнопкой мыши на проекте приложения в обозревателе решений и выбрав в появившемся меню команду Store ▸ Create App Package (Магазин ▸ Создать пакеты приложения). Когда появится первое диалоговое окно, вам нужно будет выбрать вариант No (Нет) при ответе на вопрос о том, хотите ли вы подготовить приложение для отправки в Магазин Windows, и щелкнуть на кнопке Next (Далее).

Следующее диалоговое окно, показанное на рис. 10.13, позволит вам указать местоположение создаваемого пакета и особенности его построения. Вариант Neutral (Нейтральная), выбранный по умолчанию, подходит для всех платформ. Вы можете указать, нужно ли включать в пакет файлы общедоступных символов. Благодаря им пользователи смогут анализировать программные сбои, но данные файлы содержат частную информацию о вашем приложении, поэтому, возможно, вы решите не включать их в пакет.

И наконец, щелкните на кнопке Create (Создать), чтобы создать пакет. Последнее диалоговое окно будет содержать ссылку на папку с пакетом и команду для запуска комплекта сертификации приложений для Windows. Перейдите в указанное местоположение файла, и вы увидите там файл и папку. Легче всего подготовить их к распространению, выделив и упаковав с помощью команды контекстного меню проводника Windows: Send To ▸ Compressed (zipped) folder (Отправить ▸ Сжатая ZIP-папка). В итоге у вас будет единственный файл, который вы сможете передать другим пользователям. Они смогут распаковать его на своих компьютерах.

Для того чтобы установить такой пакет на целевом устройстве, вы должны открыть его папку, щелкнуть правой кнопкой мыши на файле PowerShell-сценария, который называется Add-AppDevPackage.ps1, и выбрать в появившемся меню команду Run with PowerShell (Выполнить с помощью PowerShell). В результате PowerShell автоматически установится на всех компьютерах, работающих под управлением Windows 8. Сценарий проведет вас через

необходимую последовательность шагов, включая, если нужно, обновление политик системы безопасности компьютера и установку сертификатов. Когда все завершится, приложение будет установлено. Результат установки учебного приложения вы можете видеть на рис. 10.14.

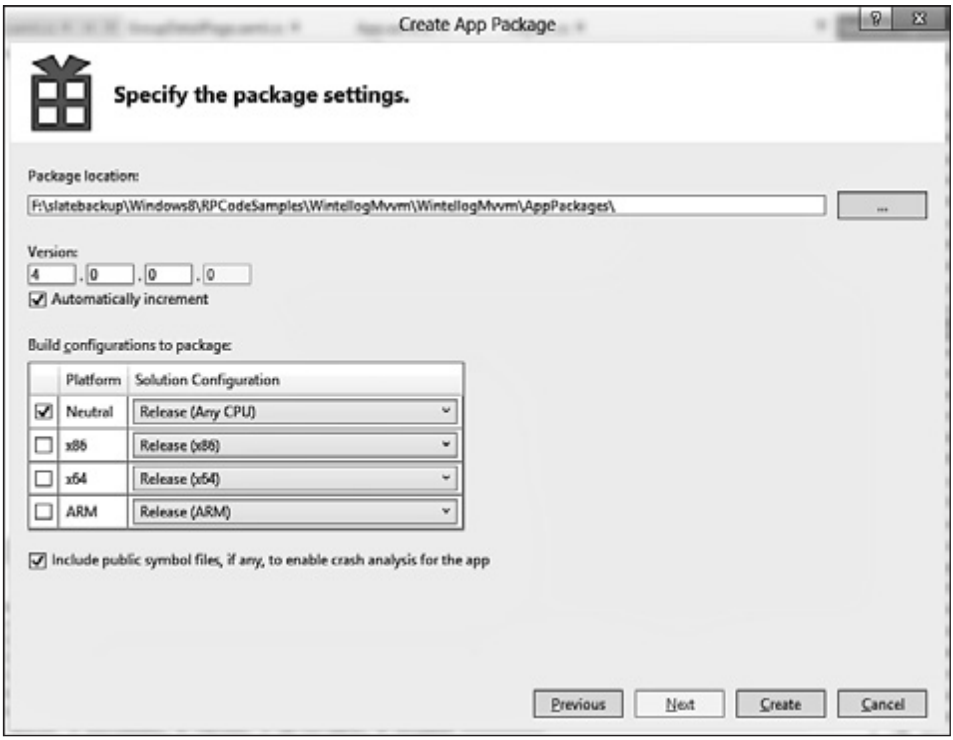


Рис. 10.13. Параметры пакетирования приложения для параллельной загрузки

Рис. 10.14. Результат запуска PowerShell-сценария установки приложения

Используя описанную процедуру, вы можете развернуть пакет приложения на нескольких целевых компьютерах, не обращаясь к Магазину Windows.

Выводы

В этой главе вы познакомились с Магазином Windows и с различными возможностями, которые он предоставляет разработчику. В частности, речь шла о различных моделях продажи приложений и о возможности реализации покупок из приложений, позволяющих активировать конкретные инструменты приложения. Я рассказал о процедуре передачи приложения в Магазин Windows, в том числе о том, как перед передачей использовать комплект сертификации приложений для Windows. Также вы узнали о том, как создавать особые пакеты, предназначенные для параллельной загрузки программных продуктов на целевые устройства без обращения к сервисам Магазина Windows.

Надеюсь, вам приятно было читать эту книгу. Мне хотелось описать все инструменты и дать все сведения, которые нужны, чтобы создать приложение и передать его для продажи в Магазин Windows. С внутренним устройством приложений разобраться довольно легко. Гораздо сложнее придумать идею программного продукта. И что бы вы ни решили создать, я желаю вам успехов в разработке приложений для платформы Windows 8.

Я буду рад получить ваши комментарии и отзывы. Если вам понравилась эта книга, я приглашаю вас посетить сайт Amazon.com и оставить отзыв о ней на странице http://bit.ly/win8design.

Это окажет неоценимую помощь другим разработчикам, которые ищут в Интернете ответы на свои вопросы. Если они смогут прочесть объективные отзывы от их соратников, это поможет им принять решение о покупке необходимой им книги. И если хотите, можете написать об этой книге и в своем блоге! Если вам хочется обсудить ее со мной или с другими разработчиками, можете посетить ее форум, который находится на сайте http://windows8applications.codeplex.com/discussions.

Это тот же сайт, на котором размещены примеры к книге. Есть там и ссылка для связи со мной. Буду рад видеть вас среди моих подписчиков в Твиттере — @JeremyLikness (черкните пару строк). Также вы можете следить за публикациями в моем блоге на странице http://csharperimage.jeremylikness.com/. Здесь вы найдете все самое свежее и актуальное из мира программирования, как и информацию о моих будущих книгах.

Спасибо!

Джереми Ликнесс

Приложения для Windows 8 на C# и XAML

Перевел с английского А. Заика

Заведующий редакцией	*А. Кривцов*
Руководитель проекта	*А. Кривцов*
Ведущий редактор	*Ю. Сергиенко*
Литературный редактор	*А. Жданов*
Художественный редактор	*Л. Адуевская*
Корректор	*С. Беляева*
Верстка	*Е. Волошина*

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 31.05.13. Формат 70x100/16. Усл. п. л. 29,670. Тираж 1500. Заказ
Отпечатано по технологии CtP в ООО «Полиграфический комплекс «ЛЕНИЗДАТ».
194044, Санкт-Петербург, ул. Менделеевская, 9. Телефон / факс (812) 495-56-10.