Microsoft

Введение в Microsoft® Silverlight™ 3

Лоуренс Морони

Здравствуйте, дорогой читатель!

Мы очень рады продолжить добрую традицию и представить Вам первую книгу о технологии Microsoft Silverlight 3 на русском языке!

Microsoft Silverlight позволяет разработчикам с помощью простых и мощных инструментов создавать интерактивные Интернет-приложения, которые становятся все популярнее среди широкого круга пользователей. Данная технология уже используется на многих известных ресурсах – как в мире, так и в России.

Silverlight 3 – важный шаг в развитии технологии. Среди новых возможностей можно отметить 3D-трансформации, аппаратное ускорение, работа вне браузера и много другое. Уже более трети Интернет-устройств в мире имеют установленный Silverlight. Принимая во внимание текущую динамику, можно с уверенностью сказать, что данная технология будет и дальше распространяться среди пользователей во всем мире, и Россия здесь не исключение.

Менее года назад мы выпустили электронную книгу «Введение в Microsoft Silverlight 2», за это время ее загрузило более 30000 читателей. Для перевода книги по Silverlight 3 мы вновь выбрали издание Лоуренса Морони, основываясь на положительных отзывах от Вас, российских разработчиков, и на международном опыте.

Книга представляет комплексный обзор Silverlight 3, который будет интересен как тем, кто уже имеет опыт работы с прошлыми версиями данной технологии, так и для начинающих свой путь в мир Silverlight.

В завершение хочу поблагодарить Вас за интерес к технологии Microsoft Silverlight и предлагаю окунуться в увлекательный мир интерактивности.

Дмитрий Халин,

Директор департамента стратегических технологий

Майкрософт Россия

ОПУБЛИКОВАНО
Microsoft Press
A Division of Microsoft Corporation
One Microsoft Way
Redmond, Washington 98052-6399
Copyright © 2009 by Laurence Moroney
Все права защищены. Ни одна часть данной книги не может быть воспроизведена в любой форме или любыми средствами без письменного разрешения издателя.
Контрольный номер Библиотеки Конгресса: 2009929278

Каталогизационная запись CIP этой книги предоставлена Британской библиотекой.

Книги Microsoft Press распространяются по всему миром оптом и в розницу. Для получения дополнительной информации о международных изданиях обращайтесь в региональный офис Корпорации Microsoft или напрямую в Microsoft Press International по факсу (425) 936-7329. Посетите наш Веб-сайт по адресу www.microsoft.com/mspress. Присылайте комментарии на адрес электронной почты mspinput@microsoft.com.

Microsoft, Microsoft Press, Access, ActiveX, DirectShow, Expression, Expression Blend, IntelliSense, Internet Explorer, MS, Outlook, Photosynth, PlayReady, Silverlight, Virtual Earth, Visual Basic, Visual Studio, Windows, Windows Live, Windows Media, Windows Server и Windows Vista являются или зарегистрированными торговыми марками или торговыми марками групп компаний Microsoft. Другие упоминаемые здесь наименования продуктов или названия компаний могут быть торговыми марками соответствующих владельцев.

Все упоминаемые в книге названия компаний, организаций, продуктов, доменные имена, адреса электронной почты, логотипы, люди, места и события являются вымышленными. Никакой связи с реальными компаниями, организациями, продуктами, доменными именами, адресами электронной почты, логотипами, людьми, местами или событиями они не подразумевают и не имеют.

Данная книга представляет видение и мнение автора. Информация, содержащаяся в ней, предоставляется без всяких выраженных, установленных или предполагаемых гарантий. Ни авторы, Microsoft Corporation, ни распространители или продавцы не несут никакой ответственности за любой ущерб, который может быть причинен прямо или косвенно данной книгой.

Body Part No. X15-88611

Всегда очень сложно подобрать слова в посвящении книги. Приходится в нескольких коротких предложениях выразить очень значимые для тебя вещи. Поэтому я просто скажу то, что очень важно для людей, которые очень важны для меня: моя жена, Ребекка, и мои дети, Клодиа и Кристофер, я посвящаю это книгу вам. Спасибо за то, что вы есть.

Также хочу поблагодарить того, кто сделал все это возможным, – Бога Авраама, Исаака, Якова и Иисуса, за то, что даровал нам жизнь, любовь, счастье и надежду

Содержание

Благодарности

Спасибо всем сотрудникам Microsoft Press и Microsoft Learning за сведение этой книги воедино. Отдельная благодарность Розмари Капертон (Rosemary Caperton) и Стиву Сэгману (Steve Sagman), в частности, которые руководили этим проектом. Я чрезвычайно признателен им за то, что были со мной откровенны и не давали выбиться из графика. Также хочу отметить выдающиеся таланты Кристины Йегер (Christina Yeager) и Пера Бломквиста (Per Blomqvist), без которых эта книга никогда не увидела бы свет!

Введение

Почему Microsoft Silverlight?

По мере роста и развития Веб, растут и ожидания Веб-пользователя. Первый Веб-браузер обеспечивал относительно простой способ создания ссылок между документами. Когда эти браузеры были объединены с протоколами межмашинного взаимодействия, появился Интернет и неожиданно документы, хранящиеся на серверах по всему миру, стали доступными друг для друга.

Со временем аудитория, использующая Интернет, изменилась. Сейчас это уже не маленькая группа людей, связанных с наукой и исследованиями в области вычислительной техники, а все население планеты в целом. И то, что было приемлемым пользовательским интерфейсом для экспертов в этой области, выглядит совершенно скудно для коммерческих приложений. Сейчас люди хотят иметь насыщенные, но при этом простые в использовании пользовательские интерфейсы, и чем больше типов информации становится доступно в Интернете (включая разнообразные виды медиафайлов), тем сложнее отвечать ожиданиям пользователей, касающимся простоты доступа к необходимым им данным.

Необходимость снабдить пользователей сложными средствами доступа к Интернет-ресурсам, которые были бы просты в использовании, приводят к появлению расширенных технологий разработки приложений. Одной из таких технологий является, например, создание «подключаемых» инструментов, которые позволяют браузеру частично использовать локальные вычислительные ресурсы пользователей.

Элементы управления ActiveX, Java-апплеты и Flash-приложения – все это примеры технологии подключаемых модулей. Асинхронный JavaScript и XML (AJAX) – еще один инструмент, созданный для разработки замечательных пользовательских интерфейсов нового поколения, в основе которых лежит концепция немедленных частичных обновлений. При использовании AJAX область экрана браузера не мигает и не блокируется, потому что полное обновление страницы выполняется редко.

AJAX позволяет разработчикам создавать более динамичные Веб-сайты, с более сложным содержимым, чем мог бы обеспечить только HTML, но его возможности тоже не безграничны. Например, он реализует асинхронный обмен информацией с сервером. Это позволяет приложениям самостоятельно обновляться в фоновом режиме и устраняет проблему с мерцанием экрана, что так часто можно наблюдать в сложных пользовательских Веб-интерфейсах. Но AJAX – это лишь механизм связи браузер-сервер. В нем нет поддержки графики, анимации, видео и других возможностей, необходимых для создания по-настоящему мультимедийных пользовательских интерфейсов.

Для выхода за рамки этих ограничений Microsoft разработана стратегия взаимодействия пользователя (UX) с Веб. В ней определяется три уровня качества взаимодействия – «хорошее», «замечательное» и «предельное», – соответственно которым принимается решение о применяемых технологиях разработки и выполнения. В данной книге эта классификация тесно связана с понятиями «насыщенный» и «насыщенность». Когда я говорю «насыщенный», я пытаюсь описать концепцию, которую сложно выразить словами. Сравните ощущение, испытываемое при работе с традиционным Веб-приложением, ограниченным возможностями браузера и HTML, с тем, когда вы работаете с настольным приложением, имеющим в своем распоряжении службы и возможности всей операционной системы. Сегодняшние Веб-приложения не обеспечивают функциональности настольных приложений, и пользователь обычно понимает, что причиной их ограниченности является технология. Применяя Silverlight (и AJAX), мы ставим цель создать Веб-приложения, очень близкие к настольным приложениям и, в конечном счете, неотличимые от них.

Самый простой уровень взаимодействия с пользователем можно реализовать в браузере, расширенном AJAX. Этот уровень определяет базовое на сегодняшний день ожидание UX – асинхронное динамическое приложение браузера, обеспечиваемое AJAX.

Верхний или «предельный» уровень – это насыщенное клиентское настольное приложение, выполняющееся в Windows Vista и использующее Windows Presentation Foundation (WPF) и .NET Framework. Эти технологии предлагают среду выполнения, которая позволяет разработчикам создавать исключительно насыщенные приложения, легко развертываемые и обслуживаемые. На этом уровне доступна передача качественной графики, видео и анимации, а также возможности приложений, такие как доступ к файловой системе и интеграция с другими настольными приложениями. Кроме того, WPF разделяет технологии дизайна и разработки, таким образом, дизайн и представление пользовательских интерфейсов создаются на новом языке программирования, называемом XML Application Markup Language (XAML). Дизайнеры получили созданные специально для них инструменты дизайна, такие как серия Microsoft Expression, и могут теперь представлять свою работу в виде XAML-документов. А разработчики претворяют в жизнь мечты дизайнеров, активируя этот XAML кодом на C#, VB.NET или JavaScript.

Я упомянул три уровня в стратегии UX, потому что по мере развития AJAX и .NET/WPF стало очевидным наличие промежуточной ниши для новой технологии, эффективно сочетающей в себе лучшие черты обоих миров – глобальную масштабируемость Интернет-приложения и насыщенность настольного приложения. Такое взаимодействие с пользователем было названо «замечательным» и обеспечивается браузером, расширенным AJAX и новой технологией, Silverlight.

Silverlight – это подключаемый модуль для браузера, формирующий визуальное представление XAML и предоставляющий программный интерфейс. Таким образом, он обеспечивает возможность совместной работы дизайнеров и разработчиков при создании Интернет-приложений, обладающих насыщенностью настольных приложений.

Первая выпущенная версия Silverlight представила JavaScript-ориентированную модель программирования, которая обеспечила мощное средство написания сценариев для XAML-элементов в браузере. Silverlight 2 существенно расширил эту модель поддержкой .NET, благодаря которой появляется возможность использования языков программирования .NET и переход на новый уровень за счет работы с XAML, обеспечения использования готовых элементов управления, поддержки работы с сетью, мощных библиотек обработки данных, расширяемости и существенного улучшения производительности.

Теперь, с выходом Silverlight 3, инструменты стали еще более мощными и разнообразными.

В данной книге рассматривается Silverlight и то, как он обеспечивает лучшее взаимодействия с пользователем в Веб. Дан обзор платформы и возможностей создания приложений, работы с мультимедиа, насыщенными изображениями и многое другое.

Silverlight 3 может изменить и изменит ваше представление о построении приложений для Веб. Вместо Веб-сайтов теперь будет создаваться Веб-впечатление. В основе замечательного впечатления лежит замечательный дизайн, а Silverlight через XAML и инструменты пакета Microsoft Expression позволяют дизайнерам и разработчикам объединить свои усилия на пути к достижению этой цели.

При написании этой книги я ставил перед собой задачу обеспечить понимание технологий, используемых совместно для разработки и развертывания Веб-приложений Silverlight, от написания базового кода для Silverlight до расширенных инструментов для создания и доставки Silverlight-содержимого. Прочитав эту книгу и проработав все примеры, вы должны быть готовы применять изученное на практике для улучшения Веб-приложений, развертываемых сейчас. А представьте, что вы сможете сделать завтра!

Для кого эта книга

Эта книга написана для разработчиков, которые готовы ежедневно работать над созданием новых и лучших Веб-приложений для пользователей Интернет и которые хотят добавить эту передовую технологию Microsoft в комплект своих знаний, чтобы понять, как с ее помощью обеспечить более интересные, функциональные и эффективные пользовательские интерфейсы. Книга написана доступно и может быть полезной также руководителям разработки для понимания места Silverlight в мире Веб-технологий Microsoft. Руководители даже могут почерпнуть из нее базовые технические сведения, которые им понадобятся, когда их разработчики с взволнованным видом придут поговорить о Silverlight. По крайней мере, они будут понимать, чем вызван такой энтузиазм!

О чем эта книга

Эта книга разделена на две части. Часть I, *Введение в Silverlight 3,* представляет основы Silverlight. В ней рассматривается, что такое Silverlight, и какие инструменты используются для создания и обслуживания Silverlight-приложений, включая Microsoft Expression Blend 3 и Microsoft Visual Studio.

В Части I также обсуждается технология XAML и то, как она использует XML для описания всего пользовательского интерфейса, начиная от компоновки элементов управления, до анимации и пр. Наконец, в этой части подробно рассматривается сам подключаемый модуль Silverlight и его взаимодействие с браузером, благодаря которому ваши приложения становятся полноправными обитателями браузера.

В Части II, *Программирование в Silverlight 3 с использованием .NET,* мы переходим к деталям высокоуровневых концепций Silverlight 3. Это обсуждение ни в коем случае не является исчерпывающим справочником, но написано как простое и строгое введение в разработку в Silverlight. Сначала рассматриваются встроенные элементы управления, им посвящены две главы, затем демонстрируется, с какой легкостью могут создаваться собственные элементы управления. Вы изучите работу с данными, взаимодействие по сети, программирование анимации, а также некоторые расширенные элементы управления для работы с мультимедиа и новые компоненты *DeepZoom* и *Photosynth*, обеспечивающие сногсшибательное представление изображений. И завершается книга обзором новой захватывающей возможности Silverlight: поддержки динамических языков программирования.

Системные требования

Для разработки Silverlight-приложений, представленных в данной книге, понадобиться следующее (опять же, все доступно на сайте *http://silverlight.net/GetStarted/):*

- Microsoft Visual Studio 2008 с пакетом обновлений SP1
- Microsoft Expression Design
- Microsoft Expression Blend 3 + SketchFlow
- Microsoft Silverlight Software Development Kit

Рекомендуемая конфигурация системы для Microsoft Silverlight: 128 МБ оперативной памяти и процессор с частотой 450 МГц или более для Windows и 1 ГБ оперативной памяти и процессор Intel с частотой 1,83 ГГц или более для Mac OSX[1].

Рекомендуемая конфигурация для Microsoft Visual Studio 2008: ЦП с частотой 2,2 ГГц или более, оперативной памяти 1024 МБ или более, экран 1280 x 1024, жесткий диск с частотой вращения 7200

[1] На Mac OSX вы не сможете использовать Visual Studio и Expression Studio (*прим. редактора*).

об/мин или выше. (Минимальные требования: ЦП с частотой 1,6 ГГц, 384 МБ оперативной памяти, экран 1024 x 768, жесткий диск с частотой вращения 5400 об/мин.) Для Windows Vista рекомендуется: ЦП с частотой 2,4 ГГц, 768 МБ оперативной памяти.

Веб-сайт компании

Данная книга представлена на Веб-сайте компании, где можно найти весь используемый в ней код. Он организован по главам. Скачать его можно с сайта компании по адресу:

http://www.microsoft.com/learning/en/us/Books/12578.aspx

Обратите внимание, данная книга основана на бета-версии Silverlight 3, поэтому не забудьте проверить обновления кода при выходе окончательной версии Silverlight.

Поддержка книги

Microsoft Press обеспечивает поддержку книг и сопроводительных материалов на следующем Веб-сайте:

http://www.microsoft.com/learning/support/books/

> **Дополнительные материалы можно найти в сети** Все новые или обновленные материалы, дополняющие эту книгу, будут размещаться на Веб-сайте Microsoft Press Online Developer Tools. К таким материалам относятся обновления содержимого книги, статьи, ссылки на сопроводительные материалы, список опечаток, примеры глав и многое другое. Этот Веб-сайт будет доступен в скором времени по адресу *www.microsoft.com/learning/books/online/developer* и будет периодически обновляться.

Вопросы и комментарии

Все комментарии, вопросы или предложения, касающиеся данной книги или сопроводительных материалов, и вопросы, ответы на которые вы не смогли найти на перечисленных выше сайтах, пожалуйста, направляйте в Microsoft Press:

mspinput@microsoft.com

Обратите внимание, что по указанному адресу не предоставляется поддержка программных продуктов Microsoft.

Глава 1

Введение в Silverlight 3

Microsoft Silverlight является следующим шагом в направлении к улучшению взаимодействия с пользователем через Веб. Цель Silverlight – обеспечить в Веб-приложениях точность отображения и качество пользовательского интерфейса (user interface, UI) доступное в настольных приложениях, чтобы Веб-разработчики и дизайнеры могли реализовывать в создаваемых решениях особые требования заказчиков. Silverlight заполняет технологический вакуум между дизайнерам и разработчикам, предоставляя им общий формат для работы. Этот формат обрабатывается браузером и основывается на Extensible Markup Language (XML), что упрощает создание шаблонов и автоматическое формирование представления. Таким форматом является XAML – Extensible Application Markup Language (расширяемый язык разметки приложений).

До появления XAML дизайнер Веб-приложений создавал дизайн, используя какой-то определенный набор инструментов и хорошо знакомую ему технологию. Затем разработчик брал то, что сделал дизайнер, и интерпретировал это с помощью другой технологии по своему выбору. Трансформировать то, что было создано дизайнером, без проблем и правильно получалось не всегда, и разработчикам приходилось вносить множество изменений, которые подчас нарушали дизайн. С Silverlight дизайнер может с помощью определенных инструментов изобразить дизайн в виде XAML, передать этот XAML разработчику, который, в свою очередь, может наполнить его функциональностью при помощи кода и развернуть.

Silverlight – это подключаемый модуль, который может использоваться в разных браузерах и на разных платформах, разработанный для обеспечения возможности создания сложных сценариев воспроизведения мультимедиа и создания насыщенных интерактивных Интернет-приложений с использованием современных технологий. Он предлагает завершенную модель программирования, которая поддерживает AJAX, .NET Framework и такие динамические языки программирования, как Python и Ruby. Возможность программирования в Silverlight 1.0 обеспечивалась поддержкой существующих Веб-технологий, включая AJAX, JavaScript и Dynamic HTML (DHTML). В Silverlight 2 добавляется поддержка динамических языков программирования и языков программирования .NET, а также появляются новые возможности, доступные только при использовании .NET Framework, такие как изолированное хранилище, работа с сетью, обширный набор элементов управления и т.д.

Данная книга представляет возможности Silverlight в целом и новшества, предлагаемые новой версией, Silverlight 3, в частности.

В первой части данной книги представлены основы Silverlight и предлагаемые инструменты для дизайна и разработки. Вторая часть книги посвящена более детальному рассмотрению модели программирования.

Silverlight и взаимодействие с пользователем

Silverlight создан как часть огромной экосистемы, используемой для обеспечения наилучшего взаимодействия с конечным пользователем. Существует ряд типовых сценариев доступа к информации через Интернет:

- Посредством мобильных устройств
- Посредством продуктов типа «Цифровой дом»
- Посредством нерасширяемых браузеров (без подключаемых модулей)
- Посредством расширяемых браузеров (использующие такие подключаемые модули, как Flash, Java или Silverlight)
- Посредством настольных приложений
- Посредством офисных программ

С годами ожидания пользователей относительно того, как должны работать эти приложения, изменились. Например, *ожидание*, что приложение на настольном компьютере должно быть более функциональным, чем приложение того же типа в мобильном устройстве. Мы как пользователи привыкли к намного большим возможностям настольных устройств по сравнению с мобильными. Кроме того, многие считают, что «так как приложение развернуто в Веб», оно может не обладать теми возможностями, которые предлагает аналогичное настольное приложение. Например, пользователь предъявляет меньшие требования к почтовому Веб-приложению, потому что не верит, что оно может предлагать те же возможности работы с электронной почтой, что и офисные программы, такие как обеспечивает Microsoft Office Outlook.

Однако по мере сближения этих платформ ожидания пользователя растут, и теперь для описания возможностей, превышающих текущий базовый уровень ожиданий, широко используется термин *насыщенный* (*rich*). Например, понятие «насыщенное Интернет-приложение» появилось в ответ на повышение уровня сложности приложений с поддержкой AJAX для обеспечения более широких динамических возможностей в таких сценариях, как работа с электронной почтой и картографическими системами. Эволюция ожиданий привела к появлению пользователей, которые хотят получать самые широкие возможности, отвечающие не только требованиям с точки зрения функциональности и эффективности приложения, но также обеспечивающие чувство удовлетворенности продуктами и службами компании, что может служить основной для установления длительных отношений между пользователем и компанией.

В результате, Microsoft направила свои ресурсы на Взаимодействие с пользователем (User Experience, UX) и поставляет инструменты и технологии, которые разработчики могут использовать для реализации UX насыщенных приложений. Кроме того, все эти средства согласованы, т.е. навыки разработки UX-ориентированных приложений будут применимы во всех областях разработки настольных и Веб-приложений. Итак, если требуется создать Веб-версию насыщенного настольного приложения, нет необходимости осваивать новые инструменты и технологии. Аналогично, если создается мобильное приложение и требуется Интернет-версия, нет необходимости иметь два набора навыков, два набора инструментов и два группы разработчиков.

На рис. 1-1 показаны доступные на сегодняшний день модели формирования представления и программирования для Веб. Как можно видеть, типовыми технологиями разработки Веб-приложений являются CSS/DHTML в модели представления и JavaScript/AJAX/ASP.NET в модели разработки. Для настольных приложений, создаваемых на .NET Framework 3.x, XAML обеспечивает модель представления, а модель разработки обеспечивается самой средой .NET. На пересечении этих

технологий – браузер с поддержкой Silverlight, обеспечивающий объединение «лучшего из обоих миров».

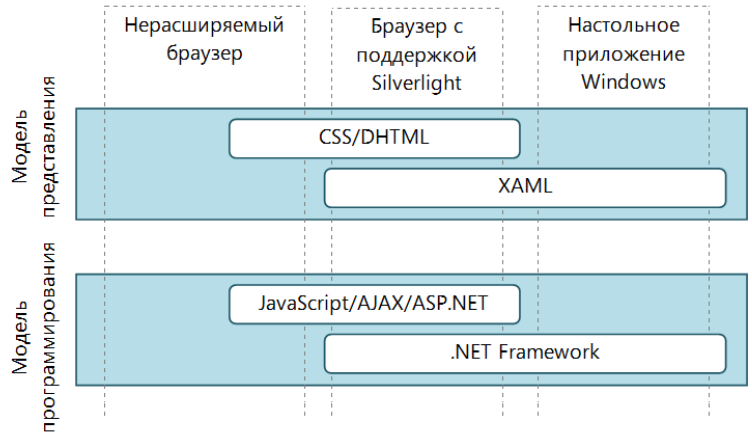

РИС. 1-1 Модели программирования и представления для Веб.

Типовое насыщенное интерактивное приложение основывается на технологиях, располагающихся в категории «нерасширяемый браузер». Типовое настольное приложение находится в другой части диапазона, для его создания используются совершенно другие технологии. Объединить эти приложения в полнофункциональное легковесное приложение, выполняющееся в браузере, позволяет браузер с поддержкой Silverlight, обеспечивающий модель дизайна CSS/DHTML и XAML и модель программирования JavaScript/AJAX/.NET Framework.

Silverlight достигает этого за счет предоставления подключаемого модуля, расширяющего функциональность браузера такими типовыми технологиями насыщенных UI, как анимация с использованием временной шкалы, векторная графика и аудиовизуальные мультимедиа. Это возможно благодаря средствам формирования визуального представления XAML для браузера, предоставляемым Silverlight. Насыщенный UI может создаваться как XAML-код, и поскольку XAML – язык, основанный на XML, и XML – это просто текст, приложение является совместимым с брандмауэром и (потенциально) удобным для поискового механизма. Браузер получает XAML и визуализирует его.

В сочетании с такой технологией, как AJAX и JavaScript, управление приложениями может быть динамическим процессом. Фрагменты XAML можно загружать и добавлять в UI, или можно редактировать, реорганизовывать или удалять XAML, находящийся в данный момент в дереве визуального представления, используя простое программирование на JavaScript.

Архитектура Silverlight

Как говорилось выше, основная функциональность Silverlight обеспечивается подключаемым модулем браузера. Этот модуль формирует визуальное представление XAML и обеспечивает модель программирования, которая может основываться либо на JavaScript, либо на .NET Framework и общеязыковой среде выполнения (common language runtime, CLR). На рис. 1-2 представлена архитектура, которая поддерживает это.

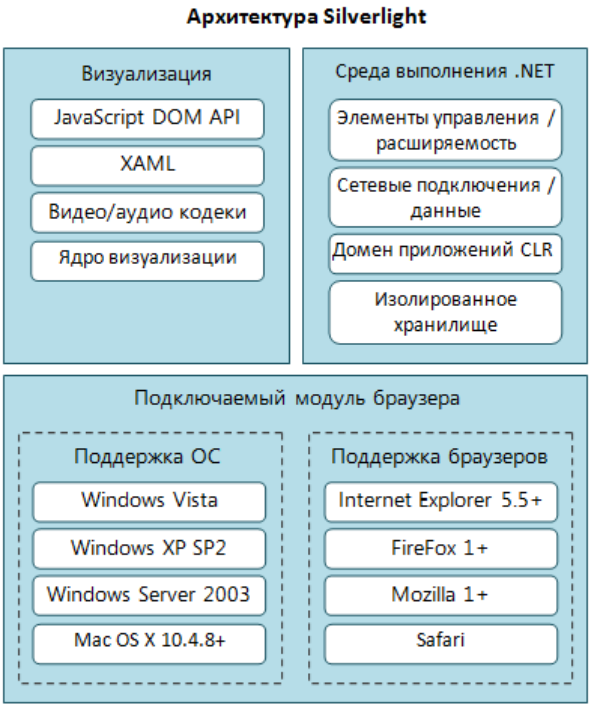

РИС. 1-2 Архитектура Silverlight.

Основным механизмом программирования поведения приложения Silverligth 1.0 в браузере является прикладной программный интерфейс (application programming interface (API) объектной модели документов (Document Object Model (DOM) JavaScript. Он позволяет перехватывать пользовательские события, возникающие в приложении (такие как перемещения мыши или щелчки определенного элемента), и вызывать код, определяющий реакцию на них. С помощью методов JavaScript DOM можно манипулировать элементами XAML, например, управлять воспроизведением мультимедиа или анимацией.

Для создания еще более насыщенного и функционального интерфейса доступны все возможности .NET Framework CLR. Помимо того, что можно сделать в JavaScript, данная возможность предлагает множество пространств имен и элементов управления, поставляемых как часть .NET Framework, что позволяет делать вещи, которые было очень сложно, или вообще невозможно, реализовать в JavaScript. Например, доступ к данным с помощью ADO.NET и LINQ, обмен информацией с Веб-сервисами, создание и использование пользовательских элементов управления и т.д.

Кроме того, среда формирования представления поставляется вместе с компонентами, необходимыми для обеспечения возможности воспроизведения в браузере файлов таких форматов, как H264, Windows Media Video (WMV), Windows Media Audio (WMA) и MP3, *без* всяких внешних зависимостей. Так, например, пользователям Macintosh не нужен проигрыватель Windows Media Player для воспроизведения WMV-содержимого, достаточно одного Silverlight. Основой всей среды формирования представления является код представления, и он управляет всем процессом визуализации. Все это встроено в подключаемый модуль браузера, поддерживающий основные браузеры Windows и Macintosh.

На рис. 1-3 представлена архитектура простого приложения, выполняющегося в браузере с использованием Silverlight.

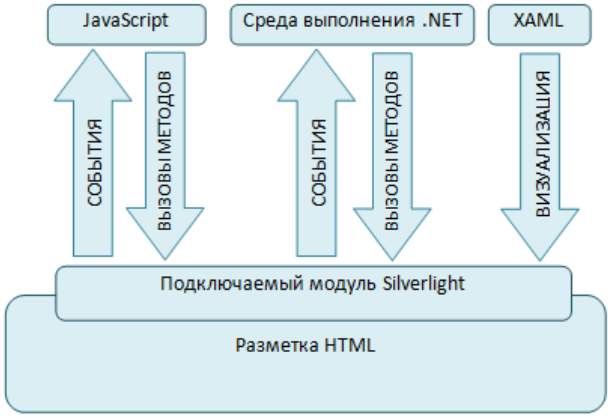

РИС. 1-3 Архитектура приложения, использующего Silverlight.

В основе выполняющегося в браузере приложения обычно лежит HTML. Эта разметка содержит вызовы для создания экземпляров подключаемого модуля Silverlight. При взаимодействии с приложением Silverlight пользователи генерируют события, которые могут быть обработаны функциями JavaScript или .NET Framework. В свою очередь, код программы может вызывать методы элементов содержимого Silverlight для управления ими, добавления нового или удаления существующего содержимого. Наконец, подключаемый модуль может читать и формировать визуальное представление XAML. Сам XAML может быть встроен в страницу, существовать как внешний статический файл или как динамический XAML, возвращаемый сервером.

Silverlight и XAML

Теперь, получив общее представление об архитектуре Silverlight и о том, как будет выглядеть типовое приложение, обратимся к основе UX: XAML.

XAML – это язык разметки на базе XML, используемый для определения визуальных элементов приложения. К ним относятся UI, графические элементы, анимации, мультимедиа, элементы управления и пр. XAML был представлен Microsoft для Windows Presentation Foundation (WPF), ранее называемом Avalon, который является технологией, ориентированной на выполнение на клиентском компьютере, и частью .NET Framework 3.0 и последующих версий. Как обсуждалось ранее, он разработан с целью заполнения технологической пропасти между дизайнерами и разработчиками при создании приложений.

XAML, используемый в Silverlight, отличается от XAML в WPF. В данном случае XAML – это *подмножество*, ориентированное на возможности и ограничения Веб-приложений. Так что те, кто знаком с XAML из WPF, заметят отсутствие некоторых тегов и функциональности, например, элемента *<Window>*.

XAML использует XML для определения UI с помощью элементов XML. Корнем всех XAML-документов Silverlight является элемент-контейнер, такой как *Canvas* (Холст), определяющий область, в которой будет отрисован ваш UI. В корневом элементе *Canvas* Веб-приложения Silverlight располагаются объявления пространств имен XML, необходимых Silverlight.

Рассмотрим пример XAML-документа в Silverlight 1.0:

```
<Canvas
  xmlns="http://schemas.microsoft.com/client/2007"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
  Width="640" Height="480"
  Background="White"
  >
</Canvas>
```

Обратите внимание, что объявляются два пространства имен. Типовой документ XAML содержит базовый набор элементов и атрибутов и дополнительный набор, для описания которого обычно используется префикс *x:*. Примером дополнительного атрибута пространства имен является широко используемый *x:Name*, с помощью которого задается имя элемента XAML, что позволяет ссылаться на него в коде. Корневой элемент *Canvas* объявляет область видимости имен всех элементов.

Элемент *Canvas* является контейнером. Это означает, что он может содержать другие элементы как дочерние. Эти элементы тоже могут быть контейнерами для других элементов, определяя UI как дерево XML-документа. Например, ниже представлен простой XAML-документ с рядом дочерних элементов, некоторые из которых сами являются контейнерами *Canvas*:

```
<Canvas
 xmlns="http://schemas.microsoft.com/client/2007"
 xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
 Width="640" Height="480"
 Background="Black"
 >
  <Rectangle Fill="#FFFFFFFF" Stroke="#FF000000"
     Width="136" Height="80"
     Canvas.Left="120" Canvas.Top="240"/>
  <Canvas>
     <Rectangle Fill="#FFFFFFFF" Stroke="#FF000000"
          Width="104" Height="96"
          Canvas.Left="400" Canvas.Top="320"/>
     <Canvas Width="320" Height="104"
          Canvas.Left="96" Canvas.Top="64">
        <Rectangle Fill="#FFFFFFFF" Stroke="#FF000000"
                Width="120" Height="96"/>
        <Rectangle Fill="#FFFFFFFF" Stroke="#FF000000"
                Width="168" Height="96"
                Canvas.Left="152" Canvas.Top="8"/>
    </Canvas>
  </Canvas>
</Canvas>
```

Здесь можно увидеть, что у корневого *Canvas* есть два дочерних элемента: *Rectangle* и другой *Canvas*. Второй *Canvas* также содержит *Rectangle* и *Canvas*, и в последнем *Canvas* находится еще два элемента *Rectangle*. Такая иерархическая структура позволяет логически группировать элементы управления и использовать для них общую компоновку и поведение.

XAML в Silverlight поддерживает ряд фигур, которые можно сочетать для получения более сложных объектов. Намного более подробно использование XAML рассматривается в Главе 4, «Основы XAML Silverlight», сейчас остановимся лишь на основных фигурах:

- ***Rectangle*** Позволяет определять на экране прямоугольник
- ***Ellipse*** Позволяет определять эллипс или круг
- ***Line*** Отрисовывает прямую линию, соединяющую две точки
- ***Polygon*** Отрисовывает многоугольник
- ***Polyline*** Отрисовывает ломаную линию
- ***Path*** Позволяет создавать нелинейные контуры (например, каракули)

Помимо этого, XAML поддерживает *кисти* (*brushes*), которые определяют, как объект закрашивается на экране. Внутренняя область объекта закрашивается с помощью кисти *fill*, а контур объекта отрисовывается кистью *stroke*. Существует множество типов кистей, включая одноцветные кисти, градиентные, изображения и видео.

Рассмотрим пример использования кисти *SolidColorBrush* (Одноцветная кисть) для закрашивания эллипса:

```
<Ellipse Canvas.Top="10" Canvas.Left="24"
        Width="200" Height="150">
    <Ellipse.Fill>
      <SolidColorBrush Color="Black" />
    </Ellipse.Fill>
</Ellipse>
```

В данном случае кисть использует один из 141 поддерживаемых Silverlight именованных цветов, *Black* (черный). Также может использоваться стандартная шестнадцатеричная RGB-запись для создания цветов.

Заливки и обводки также могут иметь градиентную заливку. Для этого используется градиентная кисть. Градиент определяется рядом контрольных точек градиента (*gradient stops*) в нормированном пространстве (*normalized space*). Так, например, если требуется определить линейный градиент, в котором цвета будут изменяться слева направо от черного к белому через оттенки серого, необходимо задать соответствующие контрольные точки на нормированной линии. В этом случае начало нормированной линии принимается за точку 0, и конец – за точку 1. Таким образом, градиент слева направо в одномерном пространстве имеет две точки градиента, 0 и 1. Если необходимо получить градиент с переходами между несколькими цветами – от черного к красному и затем к белому, например, – пришлось бы задать третью точку где-то между 0 и 1. Однако следует помнить, что при создании заливки мы работаем в двумерном пространстве, поэтому верхний левый угол представляет точка (0,0), и нижний правый угол представляет точка (1,1). Таким образом, для заливки прямоугольника с помощью градиентной кисти использовалась бы *LinearGradientBrush* следующим образом:

```
<Rectangle Width="200" Height="150" >
  <Rectangle.Fill>
    <LinearGradientBrush StartPoint="0,0" EndPoint="1,1">
      <LinearGradientBrush.GradientStops>
        <GradientStop Color="Red" Offset="0" />
        <GradientStop Color="Black" Offset="1" />
      </LinearGradientBrush.GradientStops>
    </LinearGradientBrush>
  </Rectangle.Fill>
</Rectangle>
```

XAML также поддерживает отображение текста. Для этого используется элемент *TextBlock*. Типовые свойства текста, такие как содержимое, тип шрифта, размер шрифта, обтекание и другие, можно изменять через атрибуты. Вот простой пример:

```
<TextBlock TextWrapping="Wrap" Width="100">
Привет, как дела?
</TextBlock>
```

Трансформация объектов в XAML осуществляется с помощью ряда трансформаций. К ним относятся:

- ***RotateTransform*** Поворачивает объект на заданный угол
- ***ScaleTransform*** Используется для изменения масштаба объекта
- ***SkewTransform*** Наклоняет объект в заданном направлении на заданный угол
- ***TranslateTransform*** Перемещает объект в направлении, соответственно заданному вектору
- ***MatrixTransform*** Используется для создания математического преобразования, которое может сочетать все вышеперечисленное

Группируя существующие трансформации, можно создать сложную трансформацию. То есть можно, например, переместить объект с помощью трансляции, изменить его размер с помощью масштабирования и при этом повернуть его. Рассмотрим пример, в котором выполняется разворот и изменение масштаба холста:

```
<Canvas.RenderTransform>
  <TransformGroup>
    <RotateTransform Angle="-45" CenterX="50" CenterY="50"/>
    <ScaleTransform ScaleX="1.5" ScaleY="2" />
  </TransformGroup>
</Canvas.RenderTransform>
```

Поддержка анимации в XAML реализована посредством определения изменений свойств со временем. Для этого используется временная шкала. Временные шкалы располагаются в *раскадровке* (*storyboard*). Существуют различные типы анимации:

- ***DoubleAnimation*** Обеспечивает возможность анимации числовых свойств, например таких, которые используются для определения положения
- ***ColorAnimation*** Обеспечивает возможность анимации цветовых свойств, таких как заливки
- ***PointAnimation*** Обеспечивает возможность анимации точек, определенных в двухмерном пространстве

Изменять свойства можно линейно, тогда свойство меняет значения с течением времени, или в режиме «ключевого кадра», когда определяется ряд контрольных точек, переходы между которыми и создают анимацию. Более подробно эти вопросы будут рассмотрены в Главе 5, «Трансформация и анимация в XAML».

Но применение XAML выходит далеко за рамки рассмотренных здесь основных понятий. С помощью XAML описываются элементы управления и компоновка UI. Боле подробно это будет рассмотрено в Главе 9, «Элементы управления Silverlight: дополнительные элементы управления», и в главах Части 2, «Разработка приложений Silverlight 3 с использованием .NET».

Silverlight и пакет инструментов Expression Studio

Microsoft Expression Studio – это современный надежный набор инструментов для дизайнеров, позволяющий им создавать артефакты, которые затем могут быть включены разработчиками при работе с пакетом инструментов Microsoft Visual Studio.

В пакет инструментов Expression Studio входит несколько продуктов:

- ***Expression Web*** Этот инструмент для Веб-разработки обеспечивает возможность использовать HTML, DHTML, CSS и другие стандартные Веб-технологии для проектирования, построения и управления Веб-приложениями.
- ***Expression Media***[1] Этот инструмент управления мультимедиа ресурсами позволяет каталогизировать и организовывать эти элементы, а также кодировать и перекодировать из формата в формат.
- ***Expression Encoder*** Это приложение разработано для обеспечения возможности управления кодированием мультимедиа ресурсов. Также может использоваться для комплектации мультимедиа Silverlight-медиаплеером в виде соответствующего кода.
- ***Expression Design*** Это инструмент графического дизайна, который можно использовать для создания иллюстраций, графических элементов и ресурсов для UI Веб- и настольных приложений.
- ***Expression Blend*** Этот инструмент обеспечивает возможность создавать UI на базе XAML для настольных приложений на WPF или Веб-приложений на Silverlight.

При работе с Silverlight можно использовать некоторые или все эти приложения. Далее в данной главе мы рассмотрим, как Expression Design, Expression Blend и Expression Encoder расширяют ваш инструментарий для проектирования и построения Silverlight-приложений.

---

[1] Expression Media исключен из Expression Studio 3 (*прим. редактора*)

Silverlight и Expression Design

Expression Design – это инструмент графического дизайна, с помощью которого можно создавать графические ресурсы приложений. Это мощный и сложный инструмент, поэтому в данной книге приводится просто обзор того, как он может применяться для Silverlight XAML.

Expression Design позволяет сочетать векторные и растровые изображения для обеспечения абсолютной гибкости. Expression Design поддерживает импорт множества форматов графических файлов, включая:

- Adobe Illustrator—PDF-совместимые (*.ai)
- Adobe Photoshop (*.psd)
- Graphical Interchange Format (.gif)
- Формат Portable Network Graphics (.png)
- Растровые изображения (.bmp, .dib, .rle)
- Форматы JPEG (.jpeg, .jpg, .jpe, .jfif, .exif)
- Windows Media Photos (.wdp, .hdp)
- Tagged Image File Format (.tiff, .tif)
- Ярлыки (.ico)

Поддерживается экспорт следующих типов изображений:

- XAML Silverlight Canvas
- XAML WPF Resource Dictionary
- XAML WPF Canvas
- Portable Document Format (.pdf)
- Adobe Photoshop (.psd)
- Tagged Image File Format (.tif, .tiff)
- Форматы JPEG (.jpeg, .jpg)
- Windows Bitmap (.bmp)
- Формат Portable Network Graphics (.png)
- Graphical Interchange Format (.gif)
- Windows Media Photos (также известный как HD Photo) (.wdp)

Как видите, Expression Design поддерживает экспорт графических ресурсов в виде XAML-файлов. Позже в данной главе будет показано, как в Expression Design разрабатывается дизайн графических элементов простого приложения. Вы будете экспортировать эти элементы в виде XAML-кода, который сможете использовать для создания приложения в Expression Blend и Visual Studio.

На рис. 1-4 представлено диалоговое окно Export (Экспорт) в Expression Design. В нем можно увидеть несколько вариантов выбора формата, один из которых – XAML Silverlight Canvas (отображается выбранным). Выбор этой опции обеспечивает форматирование рисунка с использованием подмножества элементов XAML, используемого Silverlight, что позволит импортировать полученный в результате XAML в Visual Studio или Expression Blend для построения Silverlight приложения.

РИС. 1-4 Экспорт XAML из Expression Design.

Содержимое экспортируется как XML-документ, включающий элемент *Canvas*, в котором содержатся элементы вашего дизайна. Рассмотрим пример (сокращенный):

```
<?xml version="1.0" encoding="utf-8"?>
<Canvas xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml" x:Name="Document">

  <Canvas x:Name="Layer_1" Width="640.219" Height="480.202" Canvas.Left="0" Canvas.Top="0">
    <Ellipse x:Name="Ellipse" Width="135" Height="161" Canvas.Left="0.546544"
      Canvas.Top="20.3998" Stretch="Fill" StrokeLineJoin="Round" Stroke="#FF000000"
      Fill="#FFFFC800"/>
    <Path x:Name="Path" Width="135.103" Height="66.444" Canvas.Left="-0.555986"
      Canvas.Top="-0.389065" Stretch="Fill" StrokeLineJoin="Round" Stroke="#FF000000"
      Fill="#FF000000" Data="..."/>
    <Path x:Name="Path_0" Width="19.4583" Height="23.9019" Canvas.Left="75.8927"
      Canvas.Top="76.1198" Stretch="Fill" StrokeLineJoin="Round" Stroke="#FF000000"
      Fill="#FF000000" Data="..."/>
    <Path x:Name="Path_1" Width="11.0735" Height="24.0564" Canvas.Left="60.473"
      Canvas.Top="106.4" Stretch="Fill" StrokeLineJoin="Round" Stroke="#FF000000"
      Fill="#FF000000" Data="..."/>
    <Path x:Name="Path_2" Width="76" Height="29.8274" Canvas.Left="31.5465"
      Canvas.Top="127.4" Stretch="Fill" StrokeThickness="7" StrokeLineJoin="Round"
      Stroke="#FF000000" Data="..."/>
    <Path x:Name="Path_3" Width="20.3803" Height="27.1204" Canvas.Left="31.2028"
      Canvas.Top="75.306" Stretch="Fill" StrokeLineJoin="Round" Stroke="#FF000000"
      Fill="#FF000000" Data="..."/>
  </Canvas>
</Canvas>
```

Теперь этот XAML можно вырезать и вставить в Expression Blend или Visual Studio, и вы можете использовать графический элемент в своем приложении.

Silverlight и Expression Blend

Expression Blend обладает возможностью создания приложений Silverlight. При запуске Expression Blend и создании нового проекта предлагается два типа Silverlight проектов:

Этими двумя типами проектов Silverlight являются:

- **Silverlight Application** Эта опция обеспечивает создание шаблонного Silverlight-приложения, включающего все необходимое для начала разработки Silverlight-приложения. Шаблон включает необходимые сборки .NET, папку *properties* (свойства) с манифестом

приложения, файл App.xaml с выделенным кодом, определяющим точки входа приложения, и базовую страницу с пустым холстом и файлом выделенного кода для него.

- **Silverlight Website** Аналогичен шаблону Silverlight-приложения, но сюда входит Веб-проект, содержащий HTML-страницу, в которую встраивается Silverlight-приложение, а также необходимые файлы JavaScript.

Проект Silverlight Website

При создании нового приложения Silverlight Website с использованием Expression Blend получаемый проект содержит стандартный HTML-файл со всем необходимым JavaScript-кодом для создания экземпляра элемента управления Silverlight, а также копию файла Silverlight.js, который является частью пакета для разработчика ПО (software development kit, SDK) Silverlight. В этом файле обрабатывается создание экземпляров и загрузка подключаемого модуля Silverlight для пользователей. Структура проекта представлена на рис. 1-5.

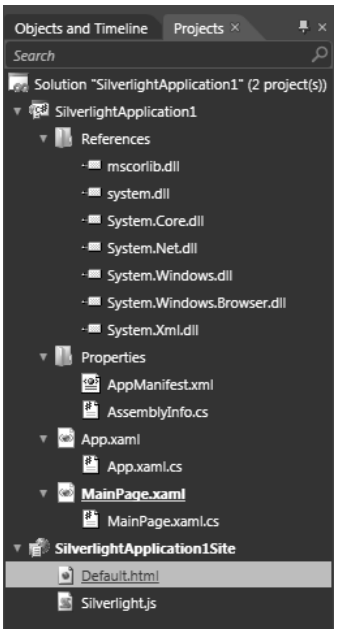

РИС. 1-5 Структура проекта приложения Silverlight Website.

Веб-страница по умолчанию

В Листинге 1-1 представлен код базовой Веб-страницы, создаваемой Expression Blend для проектов Silverlight.

ЛИСТИНГ 1-1 Файл Default.html из шаблона Silverlight

```
<div id="silverlightControlHost">
<object data="data:application/x-silverlight," type="application/x-silverlight-2" width="100%" height="100%">
  <param name="source" value="ClientBin/SilverlightApplication1.xap"/>
  <param name="onerror" value="onSilverlightError" />
  <param name="background" value="white" />
  <param name="minRuntimeVersion" value="2.0.31005.0" />
  <param name="autoUpgrade" value="true" />
  <a href="http://go.microsoft.com/fwlink?LinkID=149156&v= 3.0.40624.0"
      style="text-decoration: none;">
    <img src="http://go.microsoft.com/fwlink/?LinkId=108181"
      alt="Get Microsoft Silverlight" style="border-style: none"/>
  </a>
</object>
<iframe id='_sl_historyFrame' style='visibility:hidden;height:0;width:0;border:0px'></iframe>
</div>
</body>
```

Создание экземпляра элемента управления Silverlight происходит в теге <*object*>. Этот объект включает ряд параметров:

- Первый параметр – источник, указывающий на файл пакета приложения Silverlight (.xap), содержащий откомпилированное Silverlight-приложение. Это также может быть ссылка на статический внешний файл, ссылка на URL службы, которая может формировать XAML, или ссылка на именованный блок сценария на странице, содержащей XAML.
- Второй параметр – параметр *onerror*, определяющий блок JavaScript на странице, который вызывается при возникновении ошибки в Silverlight-приложении.
- Третий параметр, *background,* определяет цвет фона элемента управления, если таковой еще не был определен.
- Четвертый параметр, *minRuntimeVersion,* используется Silverlight для контроля версий Silverlight и определяет версию, минимально необходимую для выполнения данного приложения. Так, например, если приложение не использует ни одну из возможностей Silverlight 3, в этом параметре можно указать версию сборки Silverlight 2 (это же сделано и в листинге), тогда пользователям при просмотре этого содержимого не будет предлагаться обновить версию Silverlight до Silverlight 3.
- Четвертый параметр – *autoUpgrade*; значение *true* этого параметра обеспечивает автоматическое обновление Silverlight до последней версии при установке. Если задано *false*, ничего не происходит.

Использование Expression Blend для построения и выполнения Silverlight-приложения

В Главе 2 «Использование Expression Blend с Silverlight» Expression Blend обсуждается более подробно. Здесь мы кратко рассмотрим использование Expression Blend для построения и выполнения простого Silverlight-приложения.

С помощью New Project Wizard (Мастер нового проекта) создадим Веб-сайт Silverlight 3 и назовем его Chapter1HelloWorld. В редакторе откроем Page.xaml и добавим в рабочую область новый текстовый блок. Выберем этот текстовый блок и щелкнем вкладку Properties (Свойства).

С помощью опций, предлагаемых на вкладке Properties, можно изменять внешний вид текстового блока, например, меняя текст и размер шрифта (рис. 1-6).

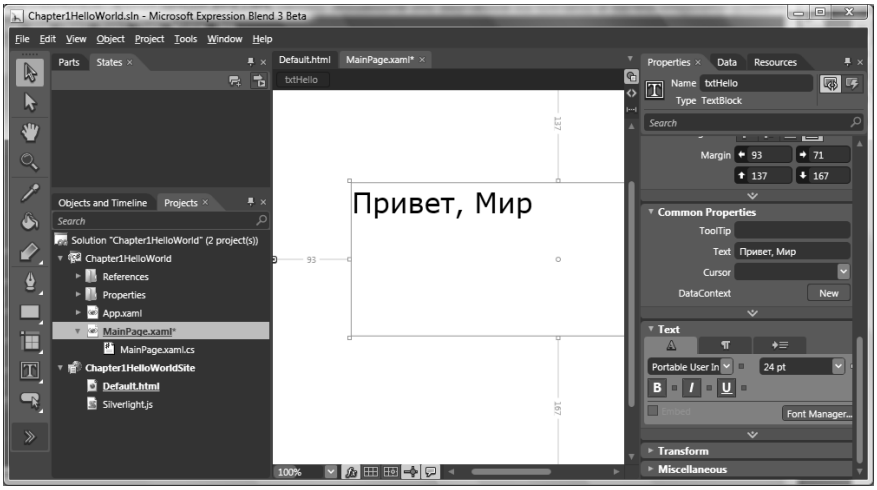

РИС 1-6 Редактирование проекта Silverlight в Expression Blend.

Также в Expression Blend 3 можно редактировать выделенный код, что было недоступно в предыдущих версиях.

Итак, например, изначально для текстового блока на рис. 1-6 не задано имя (свойство *Name* (Имя) имеет значение <No Name>(без имени)). Измените это значение на txtHello и затем откройте файл MainPage.xaml.cs. Вы увидите примерно такой C#-код:

```
public MainPage(){
  // Необходимо для инициализации переменных
  InitializeComponent();
}
```

Добавьте следующую строку под *InitializeComponent()*:

```
txtHello.Text += ", из Expression Blend";
```

Теперь по нажатию F5 Expression Blend откомпилирует и упакует .NET-код и затем откроет браузер. Браузер сформирует визуальное представление содержимого Silverlight, которое будет представлять собой строку 'Hello World from Expression Blend' (Привет, Мир, из Expression Blend).

Содержимое из дизайнера 'Привет, Мир' было дополнено текстом ', из Expression Blend' из .NET-кода.

Это предельно простой пример, но, надеюсь, он разбудил ваш интерес и желание познакомиться со многими разнообразными возможностями, доступными с Silverlight!

Silverlight и Expression Encoder

Expression Encoder может использоваться для кодирования, улучшения и публикации видеосодержимого с использованием Silverlight. Expression Encoder поставляется с UI, единообразным по стилю, виду интерфейса и управлению с остальными приложениями Expression Studio, или с интерфейсом командной строки, который может использоваться для обработки в пакетном режиме. Expression Encoder показан на рис. 1-7.

РИС. 1-7 Expression Encoder.

Expression Encoder позволяет импортировать видео из любого формата, для которого в вашей системе доступен и установлен фильтр DirectShow. После этого Expression Encoder перекодирует видео в VC-1 WMV, используя один из множества предустановленных профилей, оптимизированных под конечного пользователя. Предустановленные профили включают настройки для устройств, а также для содержимого, доставляемого через Интернет путем потоковой передачи или по запросу.

Вы не ограничены возможностями, предлагаемыми предустановленными профилями. Все параметры кодирования аудио- и видеоданных можно переопределить. На рис. 1-8 представлен пример настройки кодирования видеоданных.

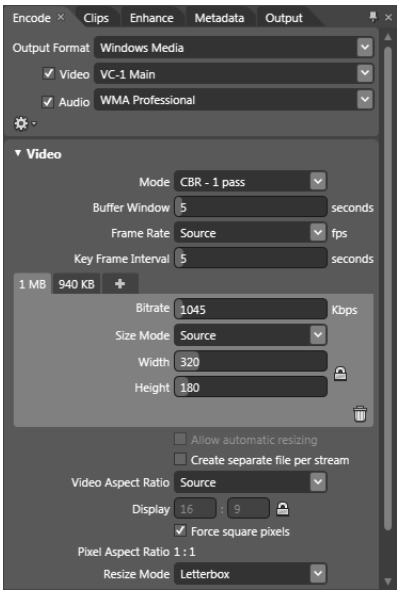

РИС. 1-8 Настройка профиля кодирования видео в Expression Encoder 3.

Expression Encoder включает ряд предустановленных медиаплееров для Silverlight. Эти приложения создадут для вашего видео Silverlight-приложение на базе JavaScript, которое может использоваться на любом Веб-сервере и обеспечит полноценный просмотр с помощью Silverlight.

Кроме кодирования, в видео могут быть добавлены метаданные. Классический вариант введения метаданных – кодирование тегов в видео, после чего приложение реагирует на эти теги. С Expression Encoder вставлять теги очень просто. Курсор воспроизведения перетягивается в необходимую точку, выбирается команда Add Marker (Добавить маркер) и вводится соответствующая информация для маркера.

Пример этому можно видеть на рис. 1-9 в правой части экрана, где для маркера заданы время и объекты, отображаемые в этот момент на экране.

На вкладке Output (Вывод) можно выбрать шаблон проигрывателя, который вы желаете использовать. На рис. 1-10 показан шаблон, соответствующий линии продуктов Expression. Чтобы создать видеоплеер с использованием этого шаблона, просто импортируйте видео, выберите этот шаблон и нажмите кнопку Encode (Кодировать).

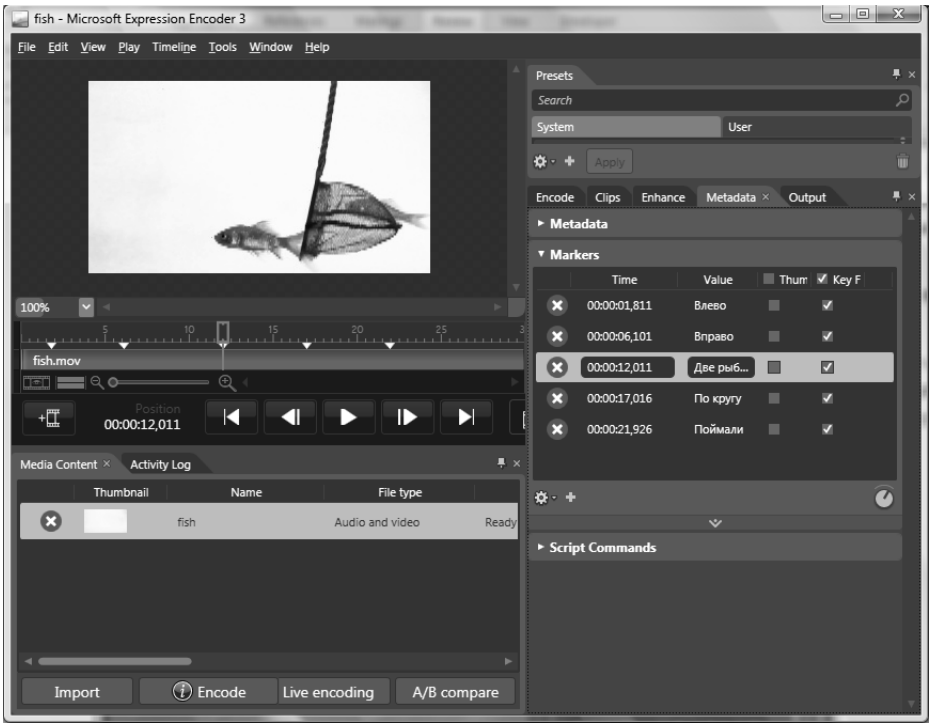

РИС. 1-9 Добавление маркеров в поток.

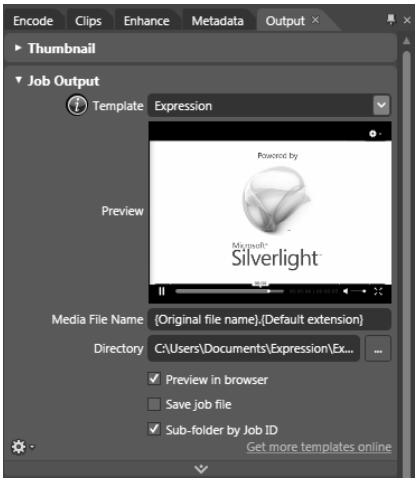

РИС. 1-10 Использование Expression Encoder 3 для создания медиаплеера Silverlight.

Выбрав шаблонный проигрыватель, вы получите полнофункциональный медиаплеер в Silverlight для своего видео. Пример медиаплеера Silverlight можно увидеть на рис. 1-11.

РИС. 1-11 Медиаплеер, сформированный Expression Encoder.

В данном разделе мы лишь слегка прикоснулись к тому, что можно сделать с помощью Expression Encoder и как он может использоваться с Silverlight. Более детальную информацию можно найти на сайте *www.microsoft.com/expression*[1].

Заключение

В данной главе представлено введение в Silverlight 3 и показана его роль в мире Веб и UX. Показано, как технология Microsoft применяется к современным сценариям UX, и дан обзор архитектуры Silverlight, включая XAML и его использование для реализации насыщенных UI.

Кроме того, показано, как Microsoft Expression Studio дополняет традиционные средства разработки, такие как Visual Studio, для создания приложений Silverlight. В частности, рассмотрено, как с помощью Expression Design создаются графические ресурсы и как они объединяются в интерактивное приложение с помощью Expression Blend, а также как можно управлять видеоресурсами, используя Expression Encoder.

Теперь пора идти дальше. В следующих нескольких главах вы больше узнаете о Silverlight API. В Главе 2 подробно рассматривается Expression Blend и его использование в Silverlight.

[1]Доступна русскоязычная версия на *www.microsoft.ru/expression* (*прим. редактора*).

Глава 2

Использование Expression Blend с Silverlight

Microsoft Expression Blend– это профессиональный дизайнерский инструмент, предназначенный для создания красивых и удобных интерфейсов настольных и Веб-приложений. Expression Blend в одном инструменте объединяет все необходимые элементы дизайна для Веб, включая видео, векторную графику, текст, анимацию, изображения, а также элементы управления. Expression Blend разработан, чтобы помочь дизайнерам создавать как настольные, так и Веб-приложения. Данная глава представляет этот инструмент, предлагая краткий тур по его возможностям. Одной главы совершенно не достаточно, чтобы охватить все многочисленные аспекты Expression Blend, но к концу этой главы вы освоите базовые понятия и будете готовы к дальнейшему изучению возможностей этого замечательного инструмента самостоятельно!

Начало работы с Expression Blend

Expression Blend доступен как часть пакета Microsoft Expression. Подробнее об этом можно узнать на сайте *http://www.microsoft.com/expression*.

После загрузки и установки Expression Blend запустите его из меню Пуск (Start). Вы увидите Интегрированную среду разработки (integrated development environment, IDE), как показано на рис. 2-1.

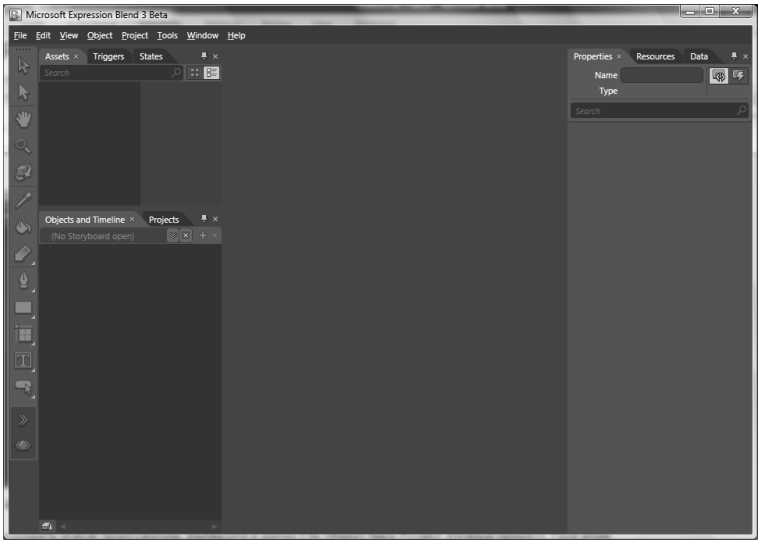

РИС. 2-1 Expression Blend IDE.

Чтобы создать новое приложение, выберите в меню File (Файл) New Project (Новый проект). При этом откроется диалоговое окно New Project, как показано на рис. 2-2.

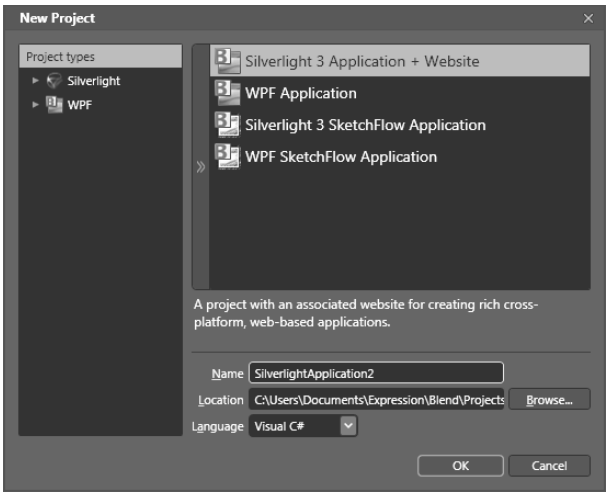

РИС.2-2 Диалоговое окно New Project в Expression Blend.

Данное диалоговое окно предлагает следующие опции:

- **Silverlight 3 Application + Website** Используйте эту опцию для создания Silverlight-приложения, содержащего файлы, необходимые для его размещения на Веб-странице.
- **WPF Application** Эта опция обеспечит создание клиентского приложения Microsoft .NET Framework, построенного на Windows Presentation Foundation (WPF).
- **Silverlight 3 SketchFlow Prototype** SketchFlow – это новая технология в Expression Blend 3, позволяющая разрабатывать прототипы приложений, которые могут быть быстро преобразованы в реальные приложения. Подробнее о SketchFlow рассказывается в этой главе далее.
- **WPF SketchFlow Prototype** Как и шаблон Silverlight SketchFlow, эта опция позволяет быстро разрабатывать прототипы клиентских приложений, построенных на WPF.

Создание приложения Silverlight

Чтобы создать приложение Silverlight, откройте диалоговое окно New Project, выберите шаблон Silverlight 3 Application + Website и задайте имя для своего нового проекта (в данном случае – TestApp). Expression Blend создаст для вас новый проект со всем необходимым для Silverlight .NET-приложения.

Структуру проекта, создаваемую Silverlight, можно увидеть на рис. 2-3. Она идентична структуре проекта, создаваемой Microsoft Visual Studio, которая будет обсуждаться подробно в Главе 3, «Использование Visual Studio с Silverlight».

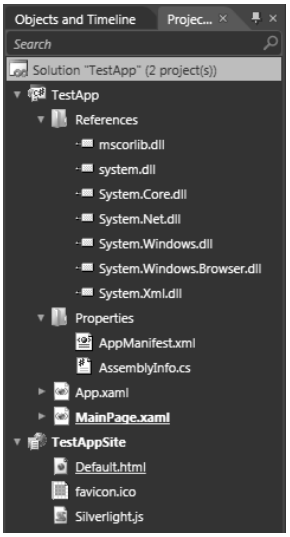

РИС. 2-3 Структура проекта Silverlight.

Обратите внимание, что создаются два файла Extensible Application Markup Language (XAML). Файл App.xaml используется для глобальных переменных, функций и параметров приложения, тогда как файл MainPage.xaml – это начальная страница приложения по умолчанию.

Файл *MainPage*

Silverlight работает с вашим XAML как с *UserControl* (Пользовательский элемент управления). Таким образом, шаблон создает стандартное XAML-содержимое приложения в виде файла MainPage.xaml. Корневым элементом данного файла, что не удивительно, является *UserControl*, а не *Canvas*, как можно было бы ожидать.

Далее представлен XAML для стандартного MainPage.xaml:

```
<UserControl
  xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
  x:Class="TestApp.MainPage"
  Width="640" Height="480">
  <Grid x:Name="LayoutRoot" Background="White"/>
</UserControl>
```

Обратите внимание, что для размещения содержимого используется <*UserControl*>.

Silverlight формирует для *MainPage* файл выделенного кода. Ему будет присвоено имя Main*Page*.xaml.cs или Main*Page*.xaml.vb в зависимости от того, какой язык программирования выбран для проекта. Файл содержит базовый код, необходимый для создания *UserControl*. Он представлен в следующем фрагменте:

```
using System;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Documents;
using System.Windows.Ink;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Animation;
using System.Windows.Shapes;

namespace TestApp
{
  public partial class MainPage : UserControl
  {
    public MainPage()
    {
      // Необходимо для инициализации переменных
      InitializeComponent();
    }
  }
}
```

Конечно, для реализации логики своего приложения вы не *ограничены* этим файлом и можете создавать другие файлы .cs (или .vb) для этого, но данный файл будет запускаться при каждом создании экземпляра элемента управления средой выполнения Silverlight. Причина этому кроется в XAML, где можно увидеть, что параметру *x:Class* задано значение *TestApp.MainPage*, указывающее на класс в который компилируется этот код.

Файл App.xaml и файлы с выделенным кодом по умолчанию

Файлы App.xaml и App.xaml.cs определяют условия запуска создаваемого приложения. Silverlight будет загружать и выполнять их первыми при запуске и закрывать последними при завершении приложения.

Для реализации этого используются события *Application_Startup* и *Application_Exit*, настроенные в шаблоне проекта. Обратите внимание, что визуальное представление *MainPage* не формируется по умолчанию, необходимо задать, чтобы он создавался при запуске приложения. Это выполняется в обработчике события *Application_Startup*, где свойство приложения *RootVisual* задается как новый экземпляр *MainPage*:

```
public App()
{

  this.Startup += this.Application_Startup;
  this.Exit += this.Application_Exit;
  this.UnhandledException += this.Application_UnhandledException;
  InitializeComponent();
}

private void Application_Startup(object sender, StartupEventArgs e)
{
  // Здесь загружается основной элемент управления
  this.RootVisual = new MainPage();
}

private void Application_Exit(object sender, EventArgs e)
{
}
```

App.xaml не поддерживает визуальные элементы напрямую, поэтому нельзя в него напрямую добавлять элементы управления или другие визуальные элементы. Не рассматривайте это как рабочую поверхность только потому, что это XAML. В данном случае, XAML используется только для определений. Например, с помощью XAML можно описать ресурсы для создаваемого приложения.

App.xaml.cs полезен для инициализации данных, которые предполагается использовать в нескольких пользовательских элементах управления. Помните об этом при разработке своего приложения. Например, можно сохранить некоторый текст, который будет использоваться приложением, объявив его ресурсом в App.xaml:

```
<Application
  xmlns="http://schemas.microsoft.com/client/2007"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
  x:Class="TestApp.App">
    <Application.Resources>
      <TextBlock x:Key="txtResource" Text="Привет"></TextBlock>
    </Application.Resources>
</Application>
```

Теперь это содержимое может быть без труда извлечено для любого элемента управления в приложении следующим образом:

```
TextBlock t = (TextBlock)Application.Current.Resources["txtResource"];
string strTest = t.Text;
```

Выполнение приложения

Особо внимательные заметят, что здесь *не хватает* страницы для размещения элемента управления Silverlight. Не волнуйтесь! Expression Blend автоматически сформирует ее для вас. Вы увидите это, когда запустите приложение.

Прежде чем двигаться дальше, добавим в *UserControl* простой *TextBlock* (Текстовый блок) для отображения некоторого текста. Вот пример:

```
<UserControl
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        x:Class="TestApp.MainPage"
        Width="640" Height="480">
```

```
    <Grid x:Name="LayoutRoot" Background="White" >
            <TextBlock Text="Привет"/>
        </Grid>
</UserControl>
```

Теперь, если выполнить это приложение (Project -> Test Solution), вы увидите на экране примерно то, что представлено на рис. 2-4.

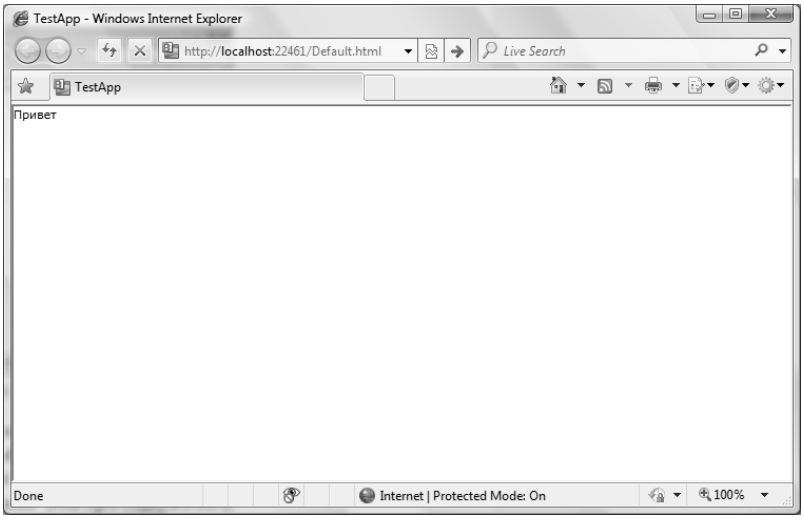

РИС. 2-4 Выполнение приложения Silverlight из Blend.

Это простое приложение выполняется с использованием облегченного встроенного Веб-сервера (иногда его называют Cassini). Веб-сервер выбирает случайный порт во время выполнения (отсюда случайный номер порта, 22461, который можно видеть в адресной строке на рис. 2-4) путем формирования HTML-страницы для размещения Silverlight-содержимого.

Посмотрим на исходный код этой страницы с помощью команды браузера View Source (Просмотр исходного кода). Обратите внимание на тег *<object>*. В нем делается попытка создания экземпляра Silverlight. В случае сбоя он формирует изображение с гипертекстовой ссылкой (HREF) на адрес, с которого можно загрузить Silverlight:

```
<div id="silverlightControlHost">
   <object data="data:application/x-silverlight,"
         type="application/x-silverlight-2" width="100%" height="100%">
     <param name="source" value="ClientBin/TestApp.xap"/>
     <param name="onerror" value="onSilverlightError" />
     <param name="background" value="white" />
     <param name="minRuntimeVersion" value="3.0.40624.0" />
     <param name="autoUpgrade" value="true" />
       <a href="http://go.microsoft.com/fwlink/?LinkID=149156"
          style="text-decoration: none;">
         <img src="http://go.microsoft.com/fwlink/?LinkId=108181"
           alt="Get Microsoft Silverlight" style="border-style: none"/>
       </a>
   </object>
   <iframe id='_sl_historyFrame' style='visibility:hidden;height:0;width:0;border:0px'></iframe>
 </div>
```

Более подробно об этом и других способах создания экземпляра объекта Silverlight рассказывается в Главе 6, «Элемент управления браузера Silverlight».

Интегрированная среда разработки Expression Blend

Expression Blend предлагает гибкую интегрированную среду разработки (IDE), которая позволяет максимально увеличить количество информации, выводимой на экран. При этом пользователь может без труда понимать, что происходит.

IDE имеет две варианта организации рабочей области приложения: рабочая область Design (Дизайн), используемая преимущественно для создания и настройки пользовательского интерфейса (UI), и рабочая область Animation (Анимация), которая предназначена, главным образом, для создания анимаций с использованием временной шкалы. Переключаться между рабочими областями можно с помощью клавиши F6, или выбирая необходимую рабочую область в опциях Active Workspace (Активная рабочая область) меню Window.

Экран в Expression Blend IDE разделен на *панели*. Каждая панель имеет собственное назначение, как вы увидите в следующих разделах.

Панель инструментов

Панель инструментов располагается на экране с левого края. Она содержит *инструменты*, такие как Selection (Выбор) или Direct Selection (Прямой выбор), которые могут использоваться для обработки любого объекта; *визуальные элементы*, такие как *Rectangle* (Прямоугольник) или *Ellipse* (Эллипс); *компоновочные элементы*, такие как *StackPanel* или *Canvas*; и *элементы управления*, такие как *Button* (Кнопка) или *TextBox* (Текстовое поле). Панель инструментов представлена на рис. 2-5.

РИС. 2-5 Панель инструментов Expression Blend.

В Expression Blend схожие инструменты могут быть сгруппированы под одним значком на панели инструментов. На рис. 2-6 видно, как можно просматривать набор подобных инструментов. Если в нижнем правом углу инструмента есть небольшой треугольник, вы можете навести курсор на этот инструмент и нажать правую кнопку мыши, чтобы увидеть всех членов «семейства» выбранного объекта. Так, например, если навести курсор на инструмент Rectangle, нажать и удерживать правую кнопку мыши, появится выпадающее окно, в котором будут представлены все остальные доступные фигуры, как показано на рис. 2-6.

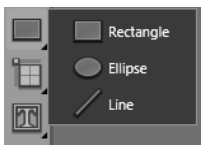

РИС. 2-6 Группировка инструментов.

Expression Blend обладает замечательным свойством – он отображает на панели инструментов значок того инструмента семейства инструментов, который вы использовали. То есть, на панели инструментов будет отображаться инструмент, который использовался последним, так что вам не придется в следующий раз удерживать правую кнопку мыши, ожидать открытия меню и выбирать этот инструмент снова.

Так, например, на рис. 2-6 на панели инструментов отображается *Rectangle*. Нажав и удерживая правую кнопку мыши, вы увидите окно с остальными доступными инструментами визуальных

элементов этого типа. Если выбрать *Ellipse* и нарисовать с его помощью что-то на рабочей поверхности, на панели инструментов будет отображаться значок *Ellipse*, а не *Rectangle*.

Панель объектов и временной шкалы

Панель Objects And Timeline (Объекты и временная шкала) представлена на рис. 2-7 и обычно располагается правее панели инструментов. Она создана с целью упрощения выполнения следующих задач:

- Просмотр всех объектов рабочей поверхности, а также их иерархии при использовании объектов-контейнеров.
- Выбор объектов для их изменения. Это не всегда возможно на рабочей поверхности, потому что объекты могут размещаться вне экрана или за другими объектами.
- Создание и изменение временных шкал анимаций. Более подробно об этом будет рассказано в разделе «Использование Expression Blend для создания анимаций» далее в этой главе.

Эта панель имеет два разных вида подсветки. Выбранный в настоящий момент элемент выделяется путем инвертирования цветов. Выделенный объект можно изменять, используя окно свойств или перетягивая его по рабочей поверхности.

Хотя на черно-белом рис. 2-7 он кажется серым, на самом деле элемент управления *LayoutRoot* (Корень компоновки) обведен цветной рамкой. На рабочей поверхности также видна эта рамка, указывающая на то, что контейнер *LayoutRoot* выбран в настоящий момент.

Кроме работы с объектами, панель Objects and Timeline используется для создания анимаций и раскадровок. Делается это путем нажатия кнопки со знаком плюс (+) вверху панели Objects and Timeline. Возможные пути использования этой панели для создания анимаций будут рассмотрены в разделе «Использование Expression Blend для создания анимаций» далее в данной главе.

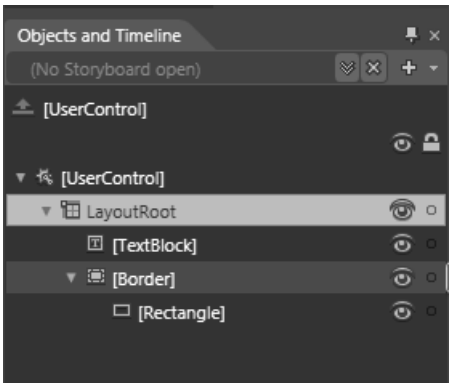

РИС. 2-7 Панель Objects And Timeline.

Рабочая поверхность

Рабочая поверхность – основная панель экрана Expression Blend IDE. Здесь можно манипулировать всеми объектами визуально или путем непосредственного изменения, лежащего в основе XAML-кода.

В правой части рабочей поверхности имеется три вкладки:

- Вкладка Design (Дизайн) открывает рабочую поверхность.
- Вкладка XAML открывает окно редактирования XAML-кода.
- Вкладка Split (Совместить) открывает разделенное окно, одна половина которого представляет рабочую поверхность, и во второй половине отображается XAML.

Рабочая поверхность в совмещенном режиме представлена на рис. 2-8.

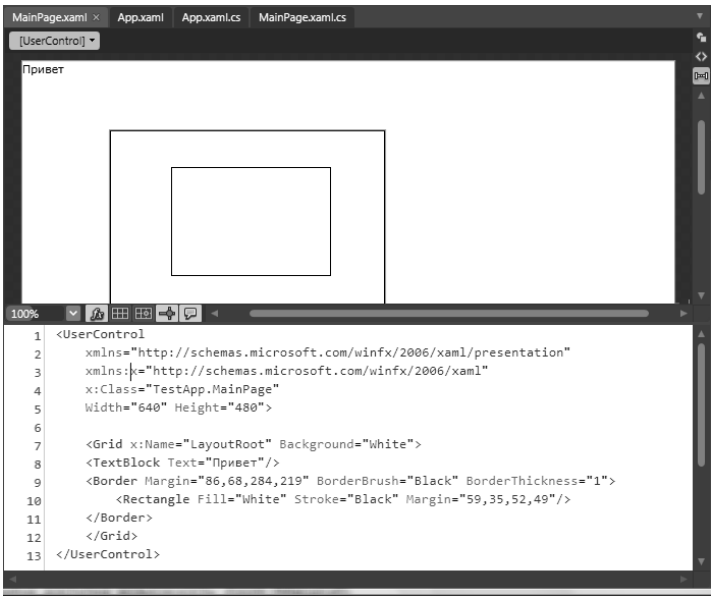

РИС. 2-8 Рабочая поверхность в совмещенном режиме.

Обратите внимание, что в режиме просмотра дизайна доступна возможность Zoom (Масштаб), поэтому при работе над сложными интерфейсами можно увеличивать масштаб для более детального просмотра и уменьшать масштаб изображения для получения общего вида. Для этого используется инструмент Zoom, располагающийся в нижнем левом углу рабочей поверхности. В выпадающем меню можно выбрать стандартный масштаб, можно ввести определенное значение в предоставленном поле или установить заданный масштаб с помощью мыши.

Панель проектов

Панель Projects (Проекты) (представленная на рис. 2-9) используется для управления файлами проекта. Что важно заметить об этой панели, это использование контекстных меню. В зависимости от того, *в каком месте* панели находится курсор в момент щелчка правой кнопкой мыши, появляются разные (но соответствующие) контекстные меню. Возможно, вам знакомы контекстные меню, предлагающие команды для определенной панели, но в данном случае разные меню выводятся по щелчку правой кнопкой мыши на решении, проекте, папке References (Ссылки) и других элементов панели Projects.

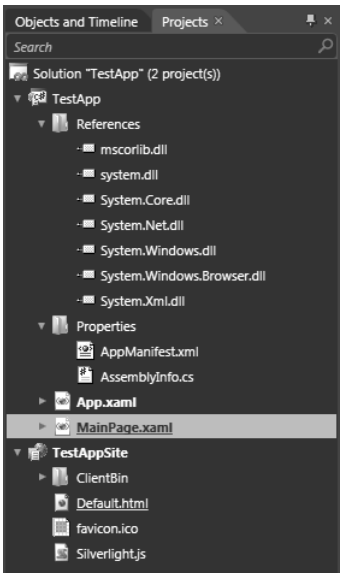

РИС. 2-9 Панель Projects.

Solution (Решение) – это коллекция из одного или более проектов. При редактировании решения им можно управлять как угодно: выполнять его сборку, отладку, очистку и управление отдельными проектами. На рис. 2-9 решение TestApp располагается вверху панели Projects, и панель показывает, что данное решение содержит два проекта.

Project (Проект) – это коллекция элементов, сочетание которых образует приложение, содержащее одну или более *страниц* Silverlight. Все ссылки на внешние компоненты, необходимые данному приложению, находятся в папке References (Ссылки) описания проекта. По правому щелчку мыши проекта появляется контекстное меню, с помощью которого можно манипулировать содержимым проекта. В контекстном меню предлагаются такие опции, как добавление новых шаблонных элементов, добавление существующих элементов из других проектов или удаление элементов из проекта.

Папка *References* в проекте используется для управления ссылками на сборки, содержащие информацию, которая необходима в создаваемом проекте. Например, если необходим специальный элемент управления, он будет скомпилирован в сборку. Тогда если вы укажете эту сборку в ссылках, вы сможете использовать ее в своем приложении.

Папка *Properties* (Свойства) содержит файл манифеста приложения, описывающий все свойства проекта, включая список ссылок, чтобы приложение понимало, откуда они загружаются во время выполнения. Не следует путать папку Properties с панелью свойств, обозначенной вкладкой Properties вверху окна, которое показано на рис. 2-9, и описываемой более подробно в следующем разделе.

Панель свойств

Панель Properties используется для управления всеми визуальными аспектами отдельного элемента. Поскольку элементы XAML имеют множество настраиваемых свойств, эта панель предлагает два очень полезных ярлыка.

Под первым ярлыком представлены свойства редактируемого элемента, сгруппированные по категориям. Обычно здесь предоставляется доступ к следующим визуальным аспектам элементов:

- **Brushes (Кисти)** Позволяет задавать для элемента заливку и обводку, а также использовать маску прозрачности. Намного более подробно использование кистей рассматривается в Главе 4, «Основы Silverlight XAML».
- **Appearance (Вид)** В разделе Appearance можно задавать расширенные свойства отображения для объекта. Обратите внимание, что доступные свойства отображения меняются в зависимости от того, какой объект редактируется в настоящий момент. Например, если редактируется элемент *Rectangle*, в разделе Appearance панели Properties вы можете задать такие характеристики, как радиус скругления углов, но при редактировании элемента *Button*, который не имеет радиуса скругления, эта опция недоступна.
- **Layout (Компоновка)** Раздел Layout позволяет редактировать различные опции компоновки для объекта, такие как Width (Ширина), Height (Высота) и опции Alignment (Выравнивание). Также опции компоновки можно использовать для изменения положения объекта в сетке, если компоновка выполняется на сетке.
- **Common Properties (Общие свойства)** Раздел Common Properties включает свойства, общие для данного *типа* объектов. Например, здесь обычно редактируются общие свойства элементов управления, не являющихся фигурами. Доступность этих опций зависит от типа редактируемого объекта. Например, если редактируется элемент управления, общим свойством является его индекс перехода по клавише Tab, но в случае редактирования фигуры индекс перехода по клавише Tab недоступен.

- **Transform (Трансформация)** Раздел Transform обеспечивает возможность редактирования свойства *RenderTransform* (Трансформация визуального представления) объекта. Это свойство определяет, как система формирования визуального отображения может изменять объект. Более подробно трансформации рассматриваются в Главе 5, «XAML: Трансформация и анимация».
- **Miscellaneous (Разное)** Раздел Miscellaneous – это место размещения всех свойств, которые не подходят ни под одну категорию.

Обратите внимание, что эти категории могут иметь подкатегории. У многих из них есть стрелка внизу, с помощью которой можно разворачивать и сворачивать панель свойств. Это позволяет скрывать свойства, которые используются реже.

Второй ярлык панели Properties – это Search (Поиск), где можно выполнять поиск определенного свойства. Например, если требуется редактировать некоторые свойства шрифта, но вы не знаете имя самого свойства, можно ввести **font** в строку поиска. Категории и доступные свойства будут отфильтрованы, и на панель будут выведены только те из них, которые имеют отношение к шрифтам. Это происходит непосредственно в процессе ввода слова поиска, поэтому если вы ищете свойство шрифта – т.е. для нашего примера, как только введены буквы **fo** – вы увидите доступные свойства, такие как ***fo****reground* и *rendertrans****fo****rm*, а также доступные свойства шрифта, как показано на рис. 2-10.

РИС. 2-10 Использование панели Properties.

Теперь рассмотрим то, как можно использовать все эти инструменты для построения приложений Silverlight.

Использование Expression Blend для создания приложений Silverlight

Expression Blend может использоваться для реализации следующих ориентированных на дизайн сценариев при создании приложения:

- Организация компоновки
- Размещение и настройка визуальных элементов

- Размещение и настройка визуальных элементов управления
- Создание анимаций

Все эти сценарии рассматриваются далее в данной главе.

Компоновка

В Silverlight для создания и организации компоновки приложения используются специальные инструменты. Доступны несколько вариантов, рассмотрим каждый из них по очереди в следующих разделах.

Использование сетки

Компоновочный элемент Grid (Сетка) позволяет раскладывать элементы в структуру, похожую на таблицу. (Не путайте компоновочный элемент Grid с элементом управления *DataGrid*, который обеспечивает функциональность, аналогичную приложению электронной таблицы.) При использовании инструмента компоновки Grid расположение своих элементов можно задавать, указывая их координаты как виртуальные строки и столбцы компоновочного элемента Grid. Например, рассмотрим следующий XAML:

```
<Grid x:Name="LayoutRoot" Background="White" >
  <Button Height="38" Margin="104,72,0,0" Width="58" Content="Кнопка"/>
  <Button Height="24" Margin="210,72,0,0" Width="54" Content="Кнопка"/>
  <Button Height="49" Margin="0,96,158,0" Width="80" Content="Кнопка"/>
  <Button Height="54" Margin="297,185,270,0" Width="67" Content="Кнопка"/>
  <Button Height="33" Margin="104,217,0,213" Width="87" Content="Кнопка"/>
</Grid>
```

Визуальное представление этого XAML будет выглядеть, как показано на рис. 2-11.

РИС. 2-11 Кнопки, расположенные случайным образом.

Теперь, для упорядочивания этих кнопок можно задать их позиции, аккуратно перетягивая их по рабочей поверхности. Но в этом случае вам пришлось бы воспользоваться инструментом масштабирования, чтобы убедиться в том, что они точно выровнены.

В качестве альтернативного варианта можно использовать компоновочный элемент Grid. Здесь местоположение кнопки на сетке задается с помощью компоновочных свойств. При создании нового Silverlight проекта, вы увидите, что файл MainPage.xaml уже имеет компоновочный элемент Grid под именем *LayoutRoot*. Выберите этот элемент на панели Objects And Timeline (Объекты и временная шкала) и посмотрите на панели Properties в разделе Layout (Компоновка), какие компоновочные свойства с ним связаны. Разверните обозреватель свойств, чтобы увидеть настройки для ColumnDefinitions (Описания столбцов) и RowDefinitions (Описания строк), как показано на рис. 2-12.

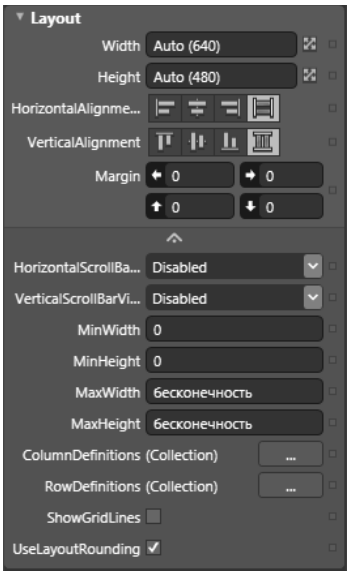

РИС. 2-12 Редактор компоновки для сетки.

ColumnDefinitions и RowDefinitions – коллекции, поэтому справа от каждого из них располагается кнопка с многоточием (…). Если щелкнуть такую кнопку, откроется другое диалоговое окно. Выберите кнопку рядом с параметром ColumnDefinitions, и на экране появится Collection Editor (редактор коллекции ColumnDefinition), как показано на рис. 2-13.

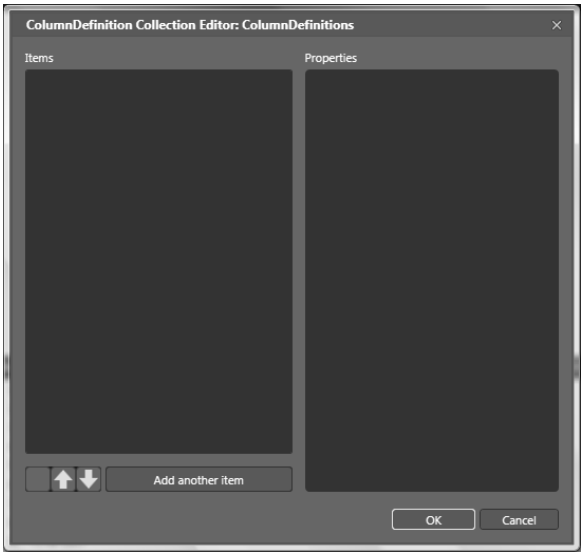

РИС. 2-13 Редактор коллекции ColumnDefinition.

Используйте это диалоговое окно для добавления, удаления и управления столбцами. Щелкните кнопку Add Another Item (Добавить еще один элемент) трижды и добавьте три столбца. Повторите аналогичные действия для параметра свойства RowDefinitions, чтобы получить сетку, состоящую из трех строк и трех столбцов. После внесения этих изменений в ColumnDefinitions и RowDefinitions на рабочей поверхности появится компоновочная сетка 3 × 3 (рис. 2-14).

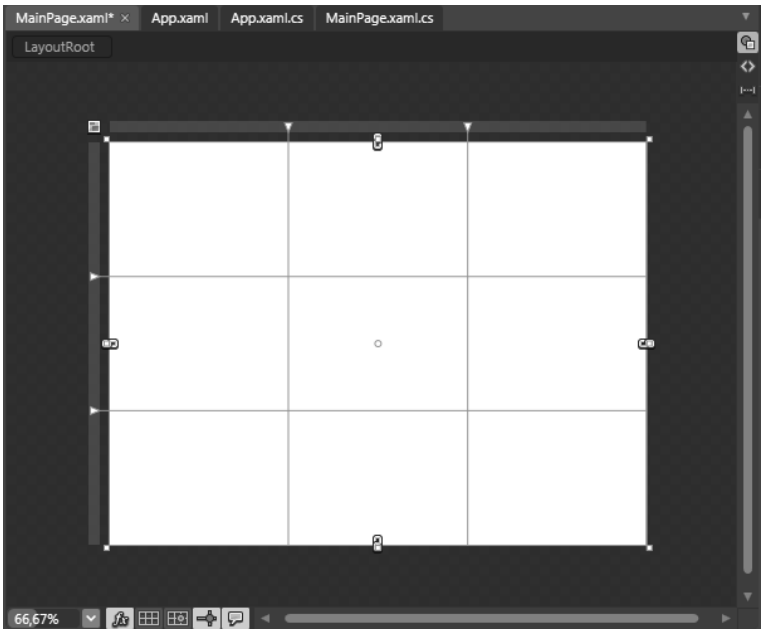

РИС. 2-14 Компоновочная сетка 3 × 3.

Теперь при размещении любого элемента на экране розовые направляющие линии будут указывать на то, как можно выполнить его привязку к определенному элементу сетки. Это показано на рис. 2-15. (На рисунке направляющие линии выглядят как более толстые серые.) Такая привязка кнопки к сетке и столбцу гарантирует, что кнопка всегда будет сохранять данные положение и размер относительно сетки.

Поместите другую кнопку в центральную ячейку сетки, как показано на рис. 2-15. На этот раз не выполняйте привязку к сетке. Затем запустите приложение и поэкспериментируйте с изменениями размеров окна[1]. Вы увидите, что относительное местоположение и размер первой кнопки остаются неизменными, тогда как вторая кнопка меняет свои ширину и/или высоту, сохраняя пропорции относительно размеров экрана.

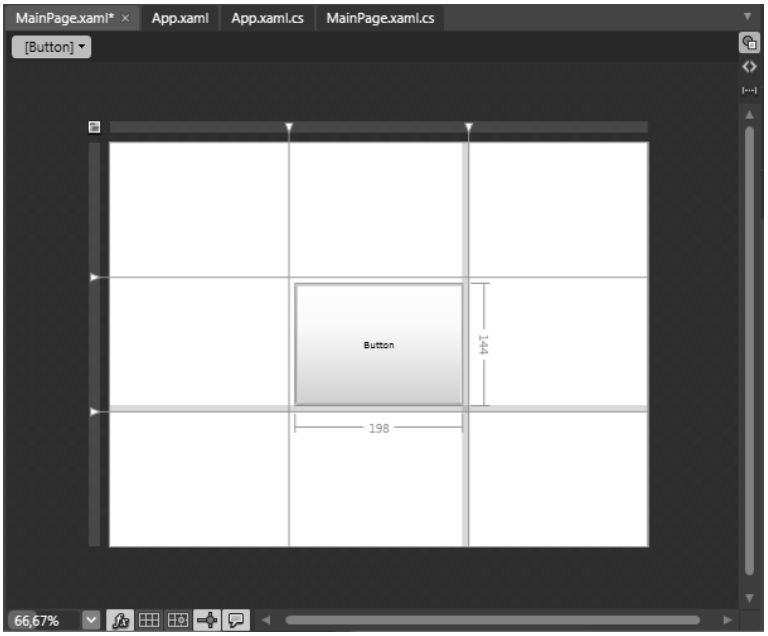

РИС. 2-15 Использование компоновки Grid.

[1] Для того чтобы при изменении размеров окна менялась и область Silverlight, необходимо свойствам Width и Height UserControl задать значение Auto (*прим. редактора*).

Использование *Canvas*

Компоновочный элемент *Canvas* является поверхностью для рисования с совершенно свободным форматом. Определить местоположение элемента управления в данном случае можно, задавая его свойства *Canvas.Top* и *Canvas.Left* или с помощью свойства *Margin* (Поля).

Например, рассмотрим следующий XAML:

```
<Canvas Height="261" Width="439">
  <Button Height="101" Width="110" Canvas.Left="101" Canvas.Top="82.5" Content="Кнопка"/>
</Canvas>
```

Как видите, свойства *Canvas.Top* и *Canvas.Left* заданы. Они показывают, что кнопка всегда будет сохранять это местоположение *относительно* верхнего левого угла родительского элемента *Canvas*. Так что, если *Canvas* будет перемещаться, кнопка тоже будет двигаться. Более подробно компоновочный элемент *Canvas* рассматривается в Главе 4.

Использование *StackPanel*

Компоновочный элемент *StackPanel* всегда выравнивает свои дочерние элементы управления в горизонтальном или вертикальном направлении, укладывая[1] их, исходя из значения свойства *Orientation* (Ориентация). Обратите внимание, что панель переопределяет позиционирование элементов управления. Например, рассмотрим следующий XAML:

```
<StackPanel Height="337" Width="224" Orientation="Vertical">
        <Button Canvas.Top="100" Height="64" Width="98" Content="Кнопка"/>
        <Button Height="85" Width="92" Content="Кнопка"/>
        <Button Height="48" Width="205" Content="Кнопка"/>
</StackPanel>
```

Как видим, свойству *Canvas.Top* первой кнопки задано значение 100. Можно было бы ожидать, что данный элемент управления будет отрисовываться в этой позиции, но, как видно на рис. 2-16, это не так. Компоновочный элемент *StackPanel* расположил эту кнопку вверху *StackPanel* (потому что свойству *Orientation* элемента *StackPanel* задано значение *Vertical* (Вертикально)).

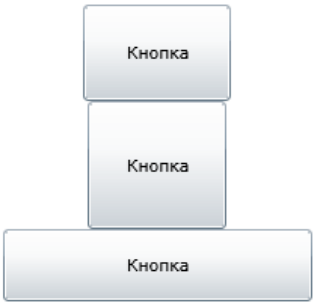

РИС. 2-16 Кнопки в *StackPanel*.

Когда в *StackPanel* располагается много элементов управления, они могут выходить за рамки самого элемента управления Panel. В этом случае элементы управления будут обрезаны границами *StackPanel*. Решить эту проблему поможет элемент управления ScrollViewer (Прокручиваемая область), который рассматривается в следующем разделе.

Использование ScrollViewer

Элемент ScrollViewer обеспечивает полосы прокрутки, благодаря которым пользователь может перемещаться по содержимому компоновки, выходящему за границы *ScrollViewer*. У *ScrollViewer* может быть только один дочерний элемент управления, поэтому обычно он используется только для размещения других контейнеров и для элементов управления с большой областью просмотра (такими как *Image*).

1По-английски «stacking», отсюда и название (*прим. переводчика*).

Например, ниже представлен *StackPanel*, содержимое которого превышает доступный размер по вертикали:

```
<StackPanel Height="300" Width="199">
  <Button Height="44" Width="86" Content="Кнопка"/>
  <Button Height="57" Width="75" Content="Кнопка"/>
  <Button Height="70" Width="59" Content="Кнопка"/>
  <Button Height="109" Width="95" Content="Кнопка"/>
  <Button Height="104" Width="88" Content="Кнопка"/>
</StackPanel>
```

В данном примере высота *StackPanel* составляет 300 пикселов, но общая высота всех кнопок – 384 пиксела. Таким образом, нижняя кнопка будет обрезана, что можно видеть на рис. 2-17.

РИС. 2-17 Элементы, не поместившиеся в *StackPanel*.

Но если поместить эту *StackPanel* в *ScrollViewer*, результат будет более приемлемым. Обратите внимание, что если не изменить высоту *StackPanel*, он по-прежнему будет обрезать кнопки. Поэтому если необходимо иметь область высотой 300 пикселов, можно задать эту высоту для *ScrollViewer*, а для *StackPanel* задать другое значение высоты. Вот пример XAML для этого:

```
<ScrollViewer Height="300" Width="300">
  <StackPanel Height="400" Width="199">
    <Button Height="44" Width="86" Content="Кнопка"/>
    <Button Height="57" Width="75" Content="Кнопка"/>
    <Button Height="70" Width="59" Content="Кнопка"/>
    <Button Height="109" Width="95" Content="Кнопка"/>
    <Button Height="104" Width="88" Content="Кнопка"/>
  </StackPanel>
</ScrollViewer>
```

То, как выглядит созданный в данном примере *ScrollViewer*, представлено на рис. 2-18.

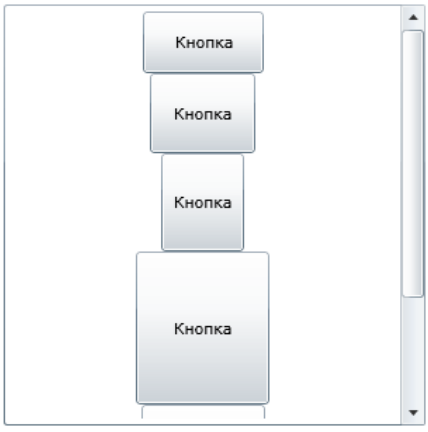

РИС. 2-18 Использование *ScrollViewer*.

Теперь можно прокручивать список кнопок вверх и вниз, и все кнопки будут доступны; ни одна из них не будет обрезана, потому что применяется *ScrollViewer*. Обратите внимание, что нижнюю кнопку (рис. 2-18) можно увидеть, если перетянуть полосу прокрутки вниз.

Элемент управления *Border*

Элемент управления *Border* используется только для отрисовки рамки, фона или того и другого вокруг элемента. Например, рассмотрим следующий XAML:

```
<Border Height="318" Width="405" Background="#FFFF0000">
  <Button Height="234"
    HorizontalAlignment="Center"
    VerticalAlignment="Center"
    Width="239"
    RenderTransformOrigin="0.5,0.5"
    Content="Кнопка">
  </Button>
</Border>
```

Это обеспечит создание красного фона за кнопкой.

Размещение и настройка визуальных элементов

Доступные визуальные элементы определены в спецификации XAML. Более подробно каждый из них рассматривается в Главе 4. А сейчас давайте перечислим основные фигуры и инструменты, предлагаемые панелью инструментов. Сюда относятся следующие фигуры:

- ***Rectangle (Прямоугольник)*** Эта фигура используется для отрисовки прямостороннего прямоугольного четырехугольника. Создав Rectangle, высота и ширина которого равны, можно получить квадрат.
- ***Ellipse (Эллипс)*** Эта фигура используется для отрисовки эллиптической формы, овала. Ее можно превратить в круг, сделав равными ширину и высоту.
- ***Line (Линия)*** Эта фигура используется для отрисовки простой прямой линии между двумя точками.

На панели инструментов также можно найти инструменты для создания фигур произвольной формы:

- **Pen (Перо)** Используется для отрисовки ряда соединенных отрезков, представленных базовым элементом *Path* (Контур).
- **Pencil (Карандаш)** Используется для отрисовки ряда соединенных элементов, которые могут быть прямыми или кривыми линиями. Expression Blend представляет отрисованные пользователем контуры с помощью базового элемента *Path*.

Все эти визуальные элементы, включая созданные с помощью инструментов Pen и Pencil, представляются одним элементом. И этот элемент может обрабатываться как любой другой объект; т.е. его можно изменять разными способами, включая задание его свойств или анимирование. Например, рассмотрим рис. 2-19, на котором с помощью инструмента Pencil отрисован ряд соединенных кривых с целью создания представления написанного от руки слова *Hello* (Привет). На панели Objects And Timeline (Объекты и временная шкала) можно увидеть объект, представленный как *Path*.

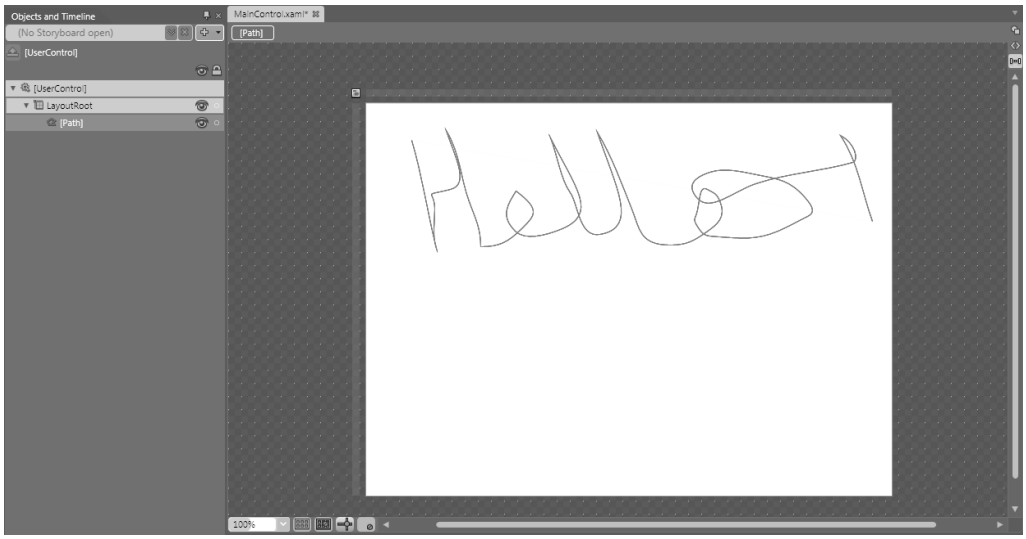

РИС. 2-19 Редактирование объекта *Path*.

Теперь слово *Hello*, написанное с помощью инструмента Pencil, рассматривается как один объект. Таким образом, вы можете редактировать его свойства, включая *Fill*, *Brush* и т.д. Просто выберите объект в представлении Objects And Timeline и внесите соответствующие изменения в панели Properties, как для любого другого объекта.

Размещение и настройка элементов управления

Expression Blend интерпретирует элементы управления абсолютно так же, как и визуальные элементы. Вы просто выбираете их на панели инструментов и размещаете на рабочей поверхности. После того, как элемент управления создан на рабочей поверхности, можно редактировать его свойства. Подробно элементы управления обсуждаются в Главе 8, «Основные элементы управления Silverlight».

Единственное, что следует заметить, – на панели инструментов Expression Blend предлагается два семейства элементов управления. Первое включает текстовые элементы управления: *TextBlock* (Текстовый блок), *TextBox и PasswordBox*. Во второе семейство входит ряд базовых элементов управления пользовательского интерфейса: *Button*, *CheckBox* (Флажок), *Combobox* (Выпадающий список), *ListBox* (Список), *RadioButton* (Переключатель), *ScrollBar* (Полоса прокрутки) и *Slider* (Ползунок).

Наконец, панель инструментов предоставляет возможность добавления элементов управления, не входящих в этот набор. Для этого необходимо выбрать ссылку Assets(Ресурсы) в нижней части панели инструментов. При этом на экран будет выведен диалог Assets, как показано на рис. 2-20.

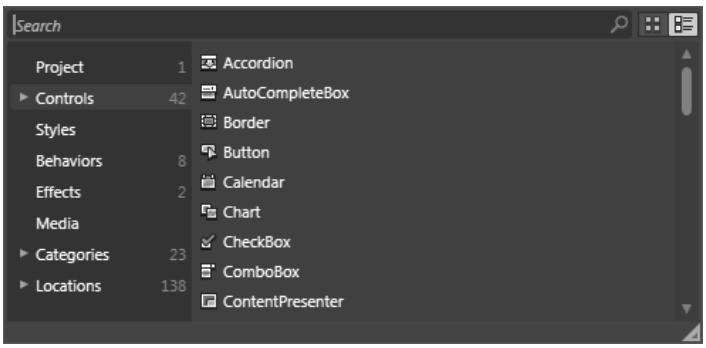

РИС. 2-20 Диалоговое окно Assets.

В диалоговом окне Assets в разделе Controls можно выбирать элементы управления и добавлять их на панель инструментов. Также можно выполнять поиск определенных элементов управления, вводя название в строку поиска. Так, например, если требуется использовать элемент управления *MediaElement* (Элемент мультимедиа), начните вводить его название в строке поиска. Увидев искомый

элемент управления (*MediaElement*, в данном случае), достаточно выбрать его, и он станет доступен на панели инструментов.

После этого элемент управления можно помещать на рабочую поверхность и работать с его свойствами в редакторе свойств, как это делается для визуальных элементов и компоновочных элементов управления.

Использование Expression Blend для создания анимаций

Подробно создание анимаций будет рассмотрено в Главе 5, но если говорить кратко, анимация создается в Silverlight за счет изменения свойства объекта во времени. Анимации такого типа можно без труда создавать визуально, используя Expression Blend и редактор временной шкалы.

Одна из форм анимации, поддерживаемых Silverlight, – *DoubleAnimation* (Анимация свойств типа Double), которая используется для изменения числовых свойств, таких как, например, ширина визуального элемента *Ellipse*. Другой тип, *ColorAnimation* (Цветовая анимация), используется для изменения цвета свойства *Brush*.

Например, рассмотрим *Ellipse*, представленный на рис. 2-21 (который выглядит как окружность, потому что его высота и ширина равны). Чтобы визуальными средствами анимировать изменение ширины элемента *Ellipse*, необходимо добавить новый элемент *Storyboard*, содержащий эту анимацию.

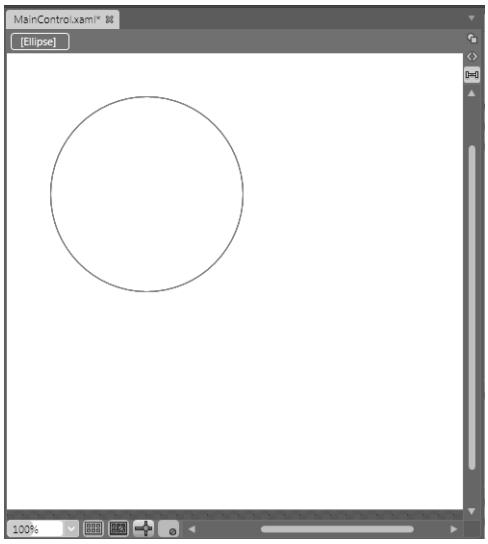

РИС. 2-21 Рисование круга.

В представлении Objects And Timeline выберите *Ellipse* и нажмите кнопку + рядом со списком *Storyboard* в верхней части панели. После принятия в диалоговом окне Create Storyboard Resource (Создание раскадровки как ресурса) настроек по умолчанию появится редактор временной шкалы. Также вверху окна Expression Blend будет выведено сообщение Storyboard1 Timeline Recording Is On (Ведется запись временной шкалы для Storyboard1). Представление Objects And Timeline можно перетащить под рабочую панель, чтобы упростить работу с временной шкалой. Экран должен выглядеть примерно так, как показано на рис. 2-22.

Найдите на временной шкале желтую линию. Она обозначает *текущее* положение курсора воспроизведения на временной шкале. Перетащите его на отметку 2 секунды и щелкните инструмент Record Keyframe (Запись ключевых кадров), располагающийся в верхней части временной шкалы. Он выглядит как овал с небольшим зеленым знаком плюс (+) снизу справа. На временной шкале на отметке 2 секунды появится маленький овал, как показано на рис. 2-23.

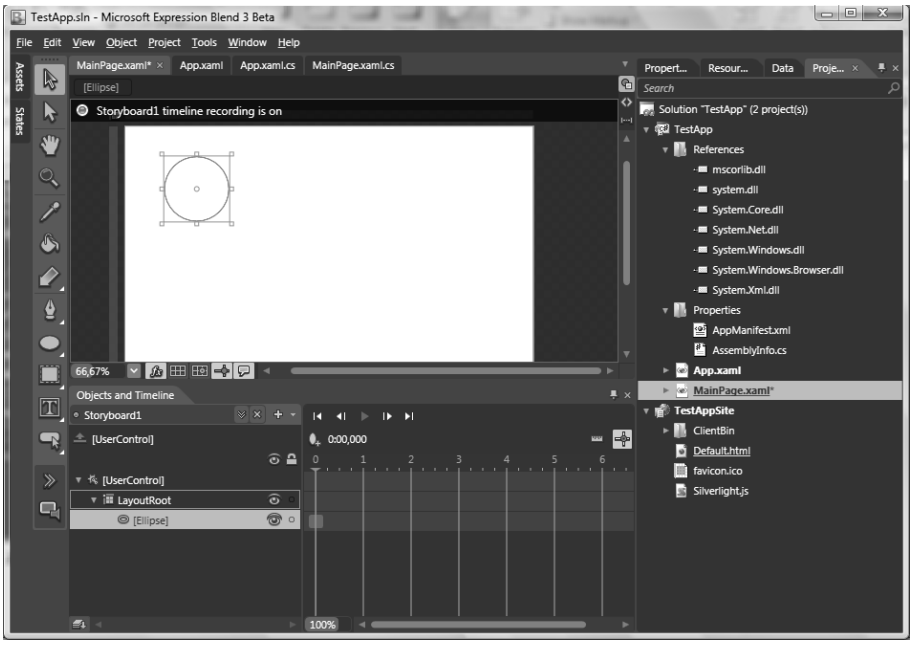

РИС. 2-22 Редактирование временной шкалы.

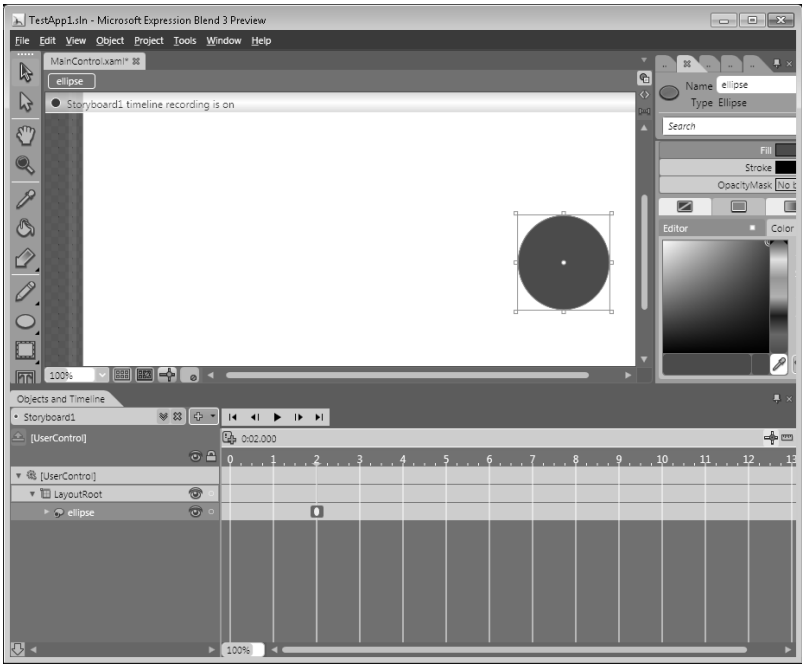

РИС. 2-23 Добавление ключевого кадра.

После того, как определен ключевой кадр, любые вносимые изменения в свойства объекта записываются в этот ключевой кадр. Продолжим и изменим ширину *Ellipse*, в то время как желтая линия остается на отметке 2 секунды, обозначая текущее положение на временной шкале. Например, зададим для ширины значение **200** и для свойства *Fill* значение **Red** (Красный).

Теперь подвигаем курсор воспроизведения (верхушка вертикально желтой линии на временной шкале) влево-вправо и увидим изменение ширины и цвета круга во времени.

А вот XAML-код, сформированный в результате создания анимации визуальными средствами:

```
<UserControl.Resources>
  <Storyboard x:Name="Storyboard1">
    <DoubleAnimationUsingKeyFrames Storyboard.TargetName="ellipse"
      Storyboard.TargetProperty="(FrameworkElement.Width)"
      BeginTime="00:00:00">
      <SplineDoubleKeyFrame KeyTime="00:00:02" Value="200"/>
    </DoubleAnimationUsingKeyFrames>
```

```
    <ColorAnimationUsingKeyFrames Storyboard.TargetName="ellipse"
     Storyboard.TargetProperty="(Shape.Fill).(SolidColorBrush.Color)"
     BeginTime="00:00:00">
     <SplineColorKeyFrame KeyTime="00:00:02" Value="#FFFF2200"/>
    </ColorAnimationUsingKeyFrames>
  </Storyboard>
</UserControl.Resources>
<Grid x:Name="LayoutRoot" Background="White">
<Ellipse Height="100" Width="100" Fill="White" Stroke="Black" x:Name="ellipse"/>
</Grid>
```

Намного более подробно структура этого XAML будет рассмотрена в Главе 5, но важно обратить внимание на экземпляр элемента *Storyboard.TargetName*, обозначающий, для какого элемента определена анимация, и экземпляр элемента *Storyboard.TargetProperty*, обозначающий изменяемое свойство. Как видим, в данном XAML присутствует две анимации: изменяющая ширину целевого объекта и изменяющая его цвет. Во время формирования визуального представления анимации Silverlight использует это описание для вычисления значений свойств в каждом кадре.

Использование SketchFlow

SketchFlow (Эскизирование) – новая технология в Expression Blend 3. Благодаря SketchFlow в Expression Blend появилась возможность создавать эскиз и прототип приложения, включая все взаимодействия. Этот инструмент позволяет разработчикам экспериментировать с различными идеями динамического взаимодействия с пользователем.

SketchFlow можно рассматривать как электронную версию создания прототипов на «бумажных салфетках», когда буквально создается примерный прототип внешнего вида и функциональности пользовательского интерфейса. Безусловно, данный прототип является элементом временного использования, потому что он может неоднократно меняться в процессе формирования приложения.

Рассмотрим создание прототипа простого приложения с использованием SketchFlow.

В Expression Blend создадим новое приложение и в диалоговом окне New Project (рис. 2-2 выше) выберем опцию Silverlight 3 SketchFlow Application. Назовем проект SFDemo.

После этого под рабочей областью должна появиться новая панель под названием SketchFlow Map (Карта процесса эскизирования). Если ее нет, убедитесь, что SketchFlow Map выбрана в меню Window, или нажмите Shift+F12.

Как выглядит панель SketchFlow Map, можно увидеть на рис. 2-24.

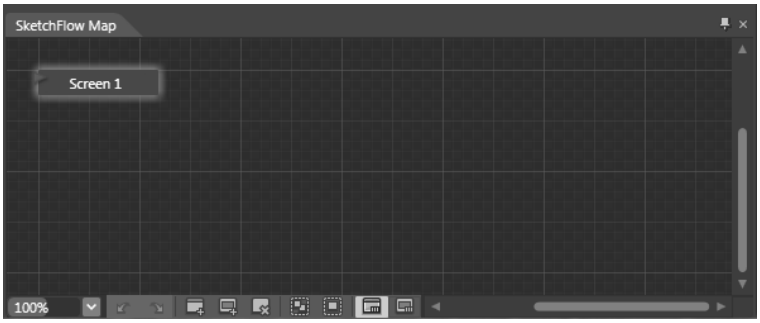

РИС. 2-24 Панель SketchFlow Map.

Ваше приложение содержит единственный XAML-документ Screen1.XAML, что визуально отражено блоком Screen 1 (Экран 1) на панели SketchFlow Map. Если провести по этому блоку указателем мыши, под ним откроется небольшое окно инструментов. В этом окне имеются опции для создания нового окна и для задания цвета блока. Протяните первую опцию Create a connected screen(Создать связанный экран) по рабочей поверхности. Отпустите кнопку мыши, и получите новое окно под

именем Screen 2 на рабочей поверхности *и* новый файл выделенного кода Screen 2.xaml в своем проекте (рис. 2-25).

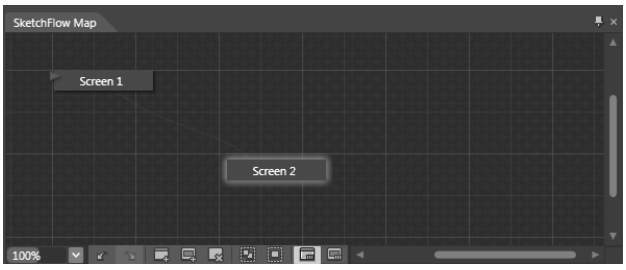

РИС. 2-25 Добавление нового окна с помощью SketchFlow.

Повторите этот процесс, чтобы разместить другое окно, связанное со Screen 1. Оно будет названо Screen 3. Далее создайте еще два окна, связанных со Screen 3. Они будут называться Screen 4 и Screen 5.

После этого рабочая поверхность должна выглядеть, как показано на рис. 2-26.

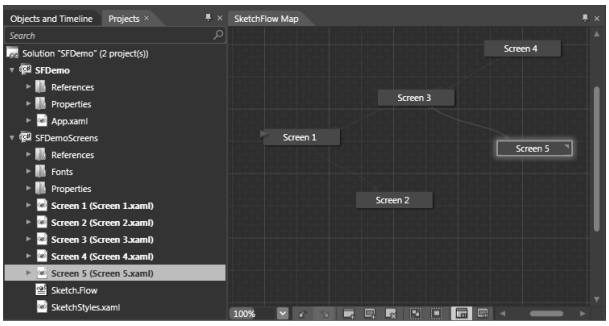

РИС. 2-26 Процесс выполнения приложения.

Теперь щелкните двойным щелчком Screen 1, чтобы открыть окно XAML Designer.

Возможно, вы заметили в своем проекте XAML-файл SketchStyles. Благодаря этому файлу можно менять внешний вид элементов управления, так чтобы они выглядели в стиле карандашного наброска.

Суть в том, чтобы было абсолютно понятно, что это прототип приложения, которое может использоваться для тестирования и демонстрации различных взаимодействий, реализующих возможные ожидания клиента.

В Assets в разделе Styles представлены различные элементы управления, предлагающие стили SketchStyles. Добавьте в окно Screen 1 пару кнопок Sketch и подпишите их Paper (Бумага) и Plastic (Пластик). Разместите над ними текстовый блок для вывода предложения пользователю о необходимости сделать выбор.

После этого окно должно выглядеть примерно, как показано на рис. 2-27.

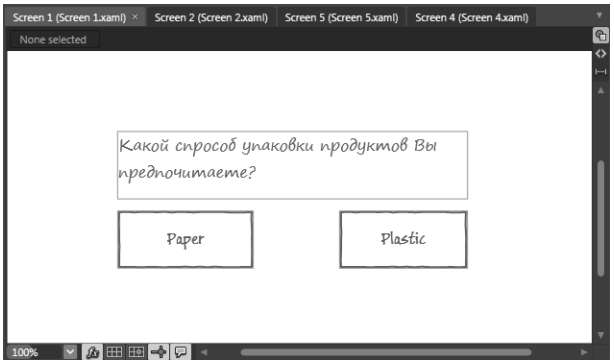

РИС. 2-27 Ваша начальная страница.

В выпадающем по щелчку правой кнопкой мыши меню любой из кнопок Sketch имеется опция Navigate To (Перейти к). Выбрав ее, вы получите список всех окон, плюс опции Back (Назад) и Forward (Вперед).

При переходе по Back или Forward SketchFlow запоминает ваше местоположение в схеме и обеспечивает перемещение назад или вперед к соответствующему окну. Однако, в этом случае, если требуется, чтобы по нажатию кнопки Paper выполнялся переход на Screen 3 и по нажатию Plastic – к Screen 2, необходимо соответствующим образом настроить свойства.

Откройте Screen 2 и добавьте некоторый текст относительно того, что «Извините, вы не можете утилизировать это», и кнопку Sketch с надписью Back, обеспечивающую переход к предыдущему окну.

Далее, откройте Screen 3. Добавьте некоторый текст относительно того, что «Как вы хотели бы утилизировать?", еще две кнопки Sketch под названием Compost (В компост) и Burn (Сжечь) и свяжите их с Screen 4 и Screen 5, соответственно.

Наконец, в этих окнах поблагодарите пользователя за утилизацию соответствующим способом и добавьте Sketch-кнопку OK, обеспечивающую возвращение к окну Screen 1. Обратите внимание, что когда вы делаете это, на поверхности SketchFlow Map отрисовывается связь от данного окна к окну Screen 1.

Построив такой базовый процесс выполнения, нажмите F5, чтобы выполнить приложение.

В браузере будет запущен проигрыватель SketchFlow Player, как показано на рис. 2-28.

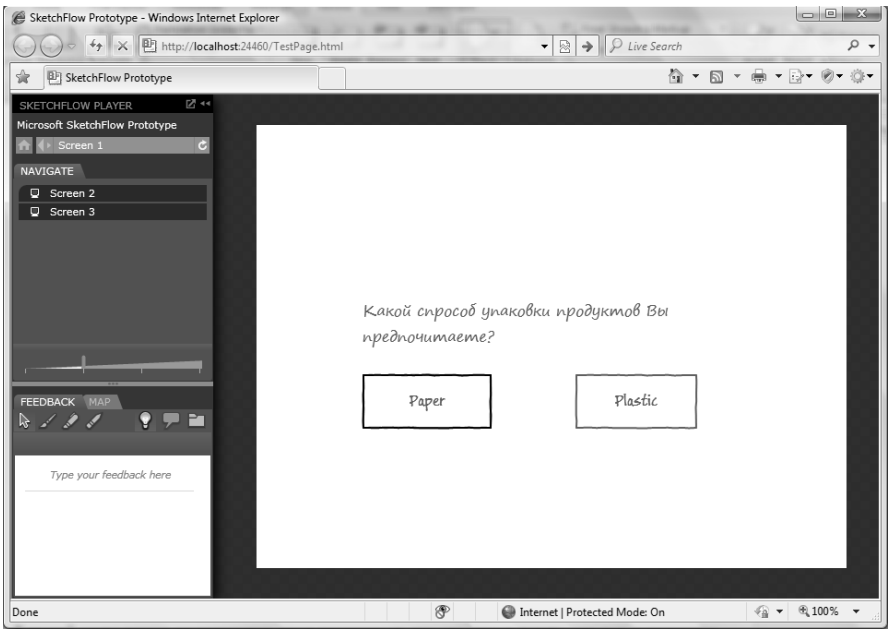

РИС. 2-28 Тестирование приложения в проигрывателе SketchFlow Player.

В правой части экрана располагается ваше приложение, так что можно видеть, как оно выполняется при выборе различных опций, таких как Paper и Plastic.

Кроме того, есть возможность переходить непосредственно к любому окну, связанному с данным, в панели Navigate (Навигация), а также оставлять отзыв и изучать процесс выполнения.

Этот краткий пример показывает лишь верхушку айсберга возможностей, предоставляемых SketchFlow. Экспериментируйте самостоятельно и вы увидите, насколько SketchFlow может быть полезен в работе с заказчиками при создании прототипов приложений.

Заключение

В данной главе были рассмотрены основы работы с Expression Blend и проведен краткий обзор возможностей, которые он предлагает дизайнеру или разработчику для создания и реализации собственных приложений Silverlight. Было показано, как можно использовать Expression Blend для разработки решений и проектов Silverlight, и какие инструменты предлагает Expression Blend IDE для создания и управления визуальными элементами, компоновкой, элементами управления и анимациями в приложении. Также дан обзор новой функциональности SketchFlow и ее использования для быстрого построения прототипов приложений.

Данная глава лишь начинает знакомство с тем, что можно делать с помощью Expression Blend, но, возможно, полученные сведения вдохновят читателей на более глубокое его изучение.

Вторая часть набора инструментов взаимодействия дизайнера и разработчика находится в Visual Studio. В Главе 3 будет рассмотрено, как можно использовать Visual Studio, что общего у нее с Expression Blend, и какие мощные возможности он предоставляет разработчикам. Будет показано, как можно создать свое первое Silverlight-приложение – головоломку с перемещением фрагментов.

Глава 3

Использование Visual Studio с Silverlight

В Главе 1, «Введение в Silverlight 3», был представлен Microsoft Silverlight и его архитектура, благодаря которой можно разрабатывать Silverlight-приложения, используя всю мощь Microsoft .NET Framework. Было показано использование расширяемого языка разметки приложений (Extensible Application Markup Language, XAML) для описания представления элементов пользовательского интерфейса (UI), взаимодействий и анимаций. Кроме того, продемонстрировано размещение Silverlight-приложения в браузере и реализация с помощью JavaScript, или размещение в .NET Runtime для браузера и реализация на C# или других языках программирования .NET.

Затем в Главе 2, «Использование Expression Blend с Silverlight», был более подробно рассмотрен Microsoft Expression Blend, инструмент для дизайна, используемый для создания приложений Silverlight.

В данной главе этот процесс будет подан с точки зрения разработчика и приведен практический пример использования Microsoft Visual Studio 2008 для построения простой игры-головоломки с перемещением фрагментов (по типу игры в пятнашки). В последующих главах этот пример будет дорабатываться. К концу данной главы вы получите прочное понимание того, как использовать C# и Visual Studio 2008 для построения .NET-приложений для Silverlight.

Установка инструментов Silverlight для Visual Studio

Пакет Microsoft Tools for Visual Studio (Visual Studio Tools для Silverlight) включают программу установки, которая обеспечивает установку среды выполнения (для операционной среды Windows), средств разработки ПО (SDK) Silverlight 3 и инструментов самой Visual Studio в правильной последовательности. Его можно загрузить по адресу *http:// silverlight.net/GetStarted/*.

Данная книга написана по бета-версии Silverlight 3. Обновления и дополнения к ней, соответственно выходящим версиям, будут доступны в моем блоге по адресу *http://blogs.msdn.com/webnext*. Обращайтесь к нему в случае возникновения каких-либо проблем.

Пожалуйста, обратите внимание, что для установки этих инструментальных средств необходимо выполнить следующие предварительные условия:

- Должна использоваться финальная версия Visual Studio 2008 с установленным пакетом исправлений SP1. Инструменты не работают с бета-версией или версией без SP1. Может использоваться любая редакция Visual Studio 2008.
- Должны быть установлены все компоненты Visual Studio для разработки Веб-сайтов.
- Должны быть удалены все предыдущие версии среды выполнения Silverlight.
- Должны быть удалены все предыдущие версии Silverlight SDK.
- Должны быть удалены все предыдущие версии Silverlight Tools для Visual Studio.

Если все эти требования выполнены, при запуске программы установки на экране появится Silverlight Tools Installation Wizard (Мастер установки инструментов Silverlight), как показано на рис. 3-1.

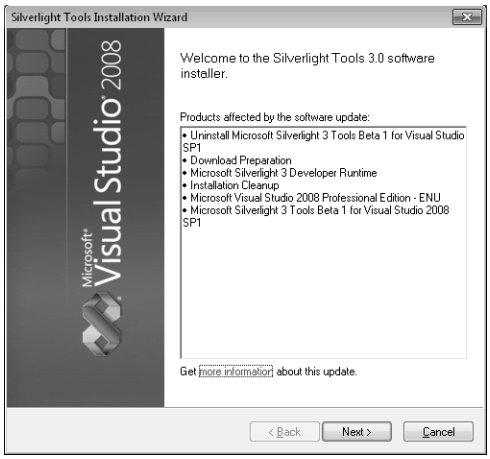

РИС. 3-1 Запуск программы установки инструментов Silverlight 3.

Щелкните Next (Далее), и появится экран с лицензионным соглашением. Сразу после принятия лицензионного соглашения начнется установка. Окно, отображаемое в процессе установки, показано на рис. 3-2.

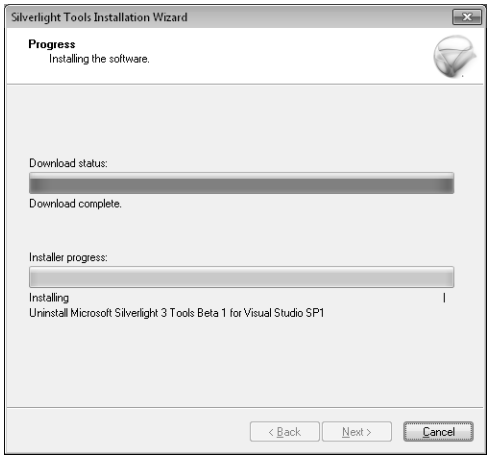

РИС. 3-2 Установка инструментов Silverlight.

Когда программа установки завершит загрузку и установку, вы получите готовые к работе среду выполнения Silverlight, SDK и Visual Studio Tools.

Использование Visual Studio для создания приложения Silverlight

Теперь, когда Silverlight Tools для Visual Studio установлены, пора научиться использовать их для проектирования и построения приложений. Рис. 3-3 показывает в действии игру-головоломку на Silverlight. Это приложение полностью написано на C# и XAML и запущено в браузере.

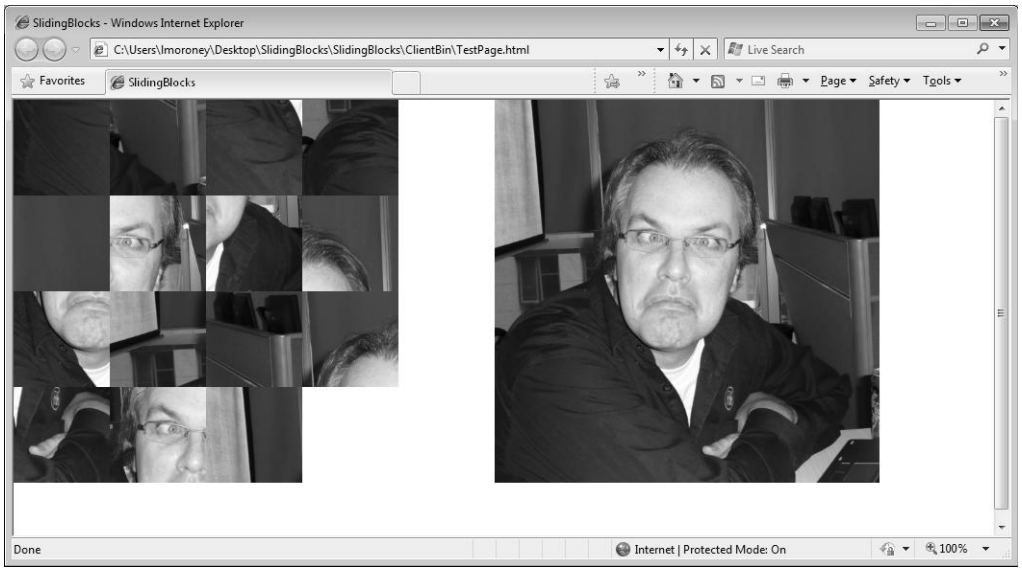

РИС. 3-3 Игра-головоломка с перемещением фрагментов.

В последующих разделах будет показано, как использовать Visual Studio 2008 и Silverlight для создания такого приложения на языке программирования C#.

Создание Silverlight-приложения в Visual Studio 2008

Установив Visual Studio, а также все необходимые инструменты и шаблоны для Silverlight, вы сможете создавать Silverlight-приложения. Для этого выберите пункт New Project в меню File. Откроется диалоговое окно New Project (рис. 3-4).

Убедитесь, что в верхнем правом углу диалога выбран пункт .NET Framework 3.5 (рис. 3-4), и в списке Project Types (Типы проекта) выберите Silverlight. Вы увидите доступные шаблоны Silverlight Application (Приложение Silverlight), Silverlight Navigation Application (Приложение Silverlight с поддержкой навигации) и Silverlight Class Library (Библиотека классов Silverlight).

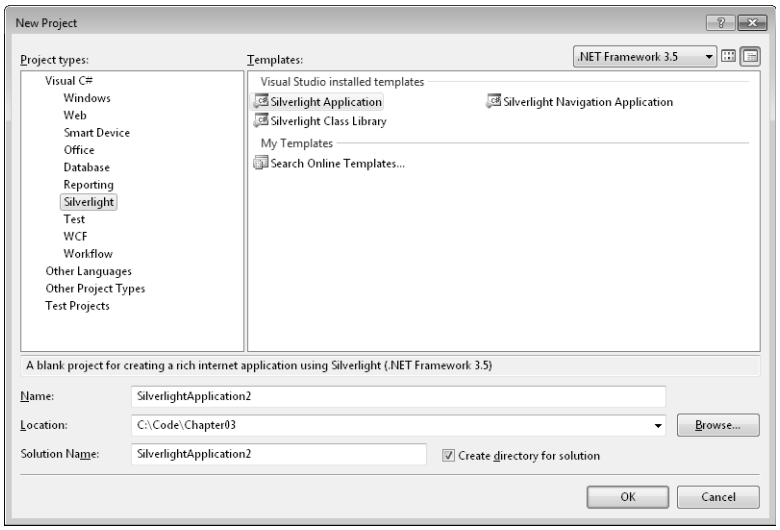

РИС. 3-4 Диалоговое окно New Project Visual Studio 2008.

Выберите шаблон Silverlight Application и задайте имя и местоположение своего проекта. Щелкните OK, и Visual Studio запустит New Silverlight Application Wizard (Мастер нового приложения Silverlight) (рис. 3-5).

Этот мастер предлагает несколько вариантов создания и управления Silverlight-приложением. Все Silverlight-приложения создаются как пользовательские элементы управления, экземпляры которых

затем могут быть созданы и размещены на странице. Таким образом, данное обсуждение сводится к тому, как это сделать.

РИС. 3-5 Мастер New Silverlight Application.

При выборе кнопки-флажка сверху мастер добавляет новый Веб-проект в решение. Этот Веб-проект может быть либо Веб-приложением ASP.NET, либо Веб-сайтом ASP.NET. Первый содержит все необходимое для развертывания приложения на Веб-сервере ASP.NET. Второй является более простой облегченной структурой, в которой страницы могут выполняться с любого сервера или даже файловой системы.

Примите настройки по умолчанию, как показано на рис. 3-5, и мастер создаст новое решение Visual Studio, содержащее элемент управления Silverlight и Веб-приложение, в котором будет размещаться этот элемент управления. Структуру проекта можно увидеть на рис. 3-6.

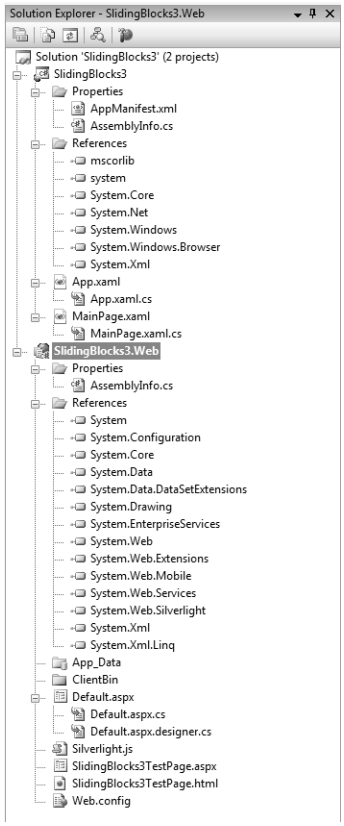

РИС. 3-6 Стандартная структура проекта Silverlight.

Как видно на рис. 3-6, было создано два решения: базовый элемент управления Silverlight (*SlidingBlocks3*) и Веб-приложение (*SlidingBlocks3.Web*). В следующем разделе вы рассмотрите эти проекты и узнаете, что они включают. После этого перейдем к созданию приложения.

Проект элемента управления Silverlight

Созданный шаблоном базовый проект содержит ряд файлов, включая манифест приложения, XAML-файл приложения с файлом выделенного кода, страницу-пример с файлом выделенного кода, информационный файл сборки и некоторые ссылки. Позднее вы можете рассмотреть каждый из этих файлов по очереди. В данном разделе представлены некоторые сложные моменты проекта Silverlight, которые, вероятно, особо нетерпеливые читатели захотят пропустить, чтобы быстрее перейти к написанию кода. Однако я рекомендую задержаться и ознакомиться с этой информацией, чтобы понимать все взаимосвязи и зависимости.

Понимание свойств проекта Silverlight

Лучше всего начать с рассмотрения свойств проекта. Для этого щелкните правой кнопкой мыши проект SlidingBlocks3 в обозревателе решения и выберите Properties. Появится диалоговое окно Project Properties (рис. 3-7).

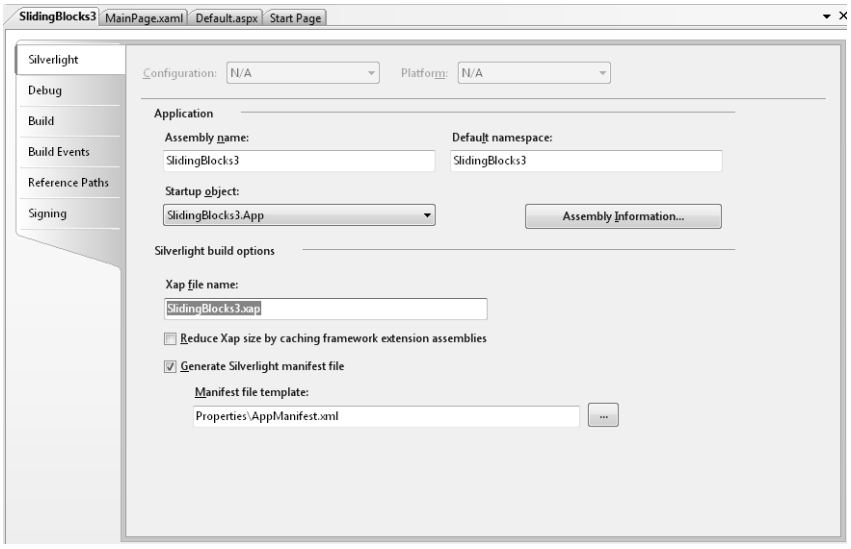

РИС. 3-7 Диалоговое окно Project Properties Silverlight.

Те, кто знаком с этой страницей, заметит, что в нем появилась дополнительная вкладка для Silverlight. Она выбрана на рис. 3-7 и представляет опции Silverlight.

Опции Assembly Name (Имя сборки) по умолчанию присваивается имя проекта. Это имя будет использоваться при компиляции приложения в DLL.

Опции Default Namespace (Пространство имен по умолчанию) по умолчанию также присваивается имя проекта. Если вы ссылаетесь на классы из этого проекта, их имена начинаются с имени этого пространства имен.

Опции Startup Object (Объект, с которого начинается запуск проекта) по умолчанию присваивается имя проекта с .App в конце (т.е. SlidingBlocks3.App). Это класс в приложении, который будет выполняться первым. Шаблон определяет этот класс в App.xaml и связанный с ним файл выделенного кода App.xaml.cs, который рассматривается в данной главе позже.

По щелчку кнопки Assembly Information (Сведения о сборке) будет вызван диалог Assembly Information (рис. 3-8). В нем можно определить метаданные своей сборки, включая Title (Название), Description

(Описание), Copyright (Авторское право) и Trademark (Торговая марка). Вся эта информация хранится в файле AssemblyInfo.cs и компилируется в приложение Silverlight.

Также доступны опции кэширования некоторых дополнительных сборок среды, позволяющие сократить размер приложения за счет того, что они не встраиваются в него, и опция формирования файла манифеста приложения. Использование манифеста приложения рассматривается в этой книге в Главе 6, «Элемент управления браузера Silverlight», при обсуждении сценариев выполнения приложений вне браузера.

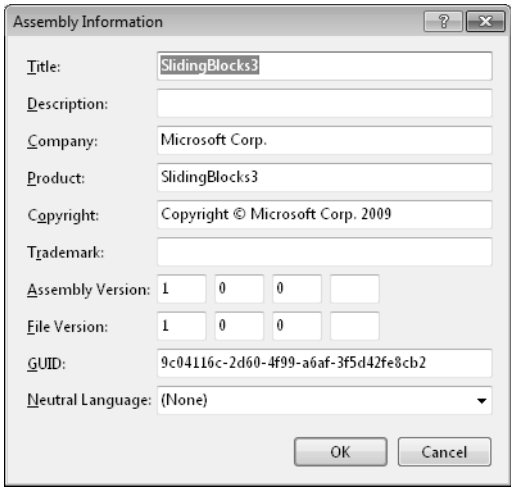

РИС. 3-8 Задание сведений о сборке.

> **Подсказка** XAP-файл – это всего лишь ZIP-файл с другим расширением. Если необходимо просмотреть его содержимое, просто переименуйте его с расширением .zip и откройте с помощью любимой утилиты для работы с архивами ZIP. Не забудьте вернуть расширение .xap, когда закончите работу с файлом!

Наконец, предлагается опция для создания самого файла манифеста Silverlight. Этот файл содержит данные по всем элементам пакета, которые будут использоваться приложением Silverlight, таким как дополнительные компоненты или элементы управления, которые необходимы для выполнения приложения. Мы еще будем возвращаться к этому по ходу книги.

Файлы свойств

Первой папкой проекта является папка Properties, содержащая файлы свойств: AppManifest.xml и AssemblyInfo.cs.

AppManifest.xml создается при компиляции проекта. Если проект имеет какие-либо зависимости во время выполнения, такие как внешние элементы управления, ссылки на них размещаются в этом файле.

Файл Assemblyinfo.cs содержит метаданные, которые должны быть скомпилированы в вашу DLL, заданную в диалоговом окне Assembly Information (рис. 3-8). Код в этом файле можно редактировать вручную, но рекомендуется использовать для этого диалоговое окно.

Ссылки

Папка References содержит ссылки на ряд сборок. Это основные сборки Silverlight, которые необходимы для запуска приложения.

- ***mscorlib*** В сборку *mscorlib* входят основные типы, используемые приложениями Silverlight.

- ***System*** В сборку *system* входят многие высокоуровневые типы, используемые для разработки и отладки приложений Silverlight, такие как компилятор и классы для отладки и диагностики.
- ***System.Core*** Сборка *System.Core* включает основные элементы управления и классы Silverlight.
- ***System.Net*** Сборка *System.Net* включает библиотеки .NET, необходимые для сетевого взаимодействия.
- ***System.Xml*** В сборку *System.Xml* входят библиотеки Silverlight для обработки XML.
- ***System.Windows*** В сборку *System.Windows* входят основные функциональные возможности Windows и Silverlight, включая элементы управления Silverlight.
- ***System.Windows.Browser*** В сборку *System.Windows.Browser* входят библиотеки, используемые для взаимодействия с браузером.

В Silverlight также имеется ряд сборок, не входящих в стандартную поставку, которые можно добавлять для обеспечения необходимой функциональности. Некоторые из них будут рассмотрены в данной книге. Примером такой функциональности является Dynamic Language Runtime (Динамическая среда выполнения).

Файлы App.xaml и App.xaml.cs

Файл App.xaml создается интегрированной средой разработки (IDE) при создании проекта Silverlight по шаблону. Обычно он используется для хранения общей информации всего приложения.

App.xaml содержит объявления, определяющие поведение приложения. Вот пример App.xaml, создаваемого шаблоном по умолчанию:

```
<Application xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        x:Class="SlidingBlocks3.App">
  <Application.Resources>
  </Application.Resources>
</Application>
```

Первое, на что следует обратить внимание, – атрибут *x:Class*, который определяет имя класса, в который будет скомпилирован этот XAML и связанный с ним файл выделенного кода. Как видите, в данном случае это SlidingBlocks3.App, который, как вы, возможно, помните из рис. 3-7, был задан на странице Project Properties как объект запуска для этого приложения. Таким образом, функциональность запуска проекта Silverlight находится в этом классе.

С помощью события *Startup* (Запуск) можно задать метод-обработчик, который будет выполняться при запуске приложения. Для этого просто указывается имя необходимой функции, описанной в файле выделенного кода. Также в событии *Exit* (Выход) можно задать имя метода-обработчика из файла выделенного кода, который будет выполняться при завершении приложения.

Далее приведен код стандартного файла выделенного кода, созданного шаблоном:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Net;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Animation;
```

```
using System.Windows.Shapes;

namespace SlidingBlocks3
{
   public partial class App : Application
   {

      public App()
      {
         this.Startup += this.Application_Startup;
         this.Exit += this.Application_Exit;

         this.UnhandledException += this.Application_UnhandledException;
         InitializeComponent();
      }

      private void Application_Startup(object sender, StartupEventArgs e)
      {
         // Загружаем основной элемент управления
         this.RootVisual = new MainPage();
      }

      private void Application_Exit(object sender, EventArgs e)
      {

      }
          private void Application_UnhandledException(object sender,
                      ApplicationUnhandledExceptionEventArgs e)
          {
            if (!System.Diagnostics.Debugger.IsAttached)
             {
             e.Handled = true;
              Deployment.Current.Dispatcher.BeginInvoke(delegate { ReportErrorToDOM(e); });
             }
           }
           private void ReportErrorToDOM(ApplicationUnhandledExceptionEventArgs e)
          {
           try
           {
            string errorMsg = e.ExceptionObject.Message + e.ExceptionObject.StackTrace;
            errorMsg = errorMsg.Replace('"', '\'').Replace("\r\n", @"\n");

            System.Windows.Browser.HtmlPage.Window.Eval("throw new Error(\"Unhandled Error
                      in Silverlight 3 Application " + errorMsg + "\");");
           }
           catch (Exception)
           {
           }
          }
   }
}
```

Сначала рассмотрим конструктор (это функция с именем, аналогичным имени модуля кода, в данном случае это *App()*). Он используется для подключения методов *Application_Startup* и *Application_Exit*. Это уже было сделано в файле XAML, поэтому нет необходимости делать это в коде. Однако это демонстрирует замечательную гибкость модели XAML/выделенный код, которая позволяет подключать события во время разработки (задавая их в XAML) или во время выполнения (объявляя их в коде).

Далее можно проверить методы *Application_Startup* и *Application_Exit*. Обратите внимание, что они принимают два параметра: объект, формирующий событие, и объект аргументов. Такая сигнатура методов очень распространена в приложениях Silverlight.

В методе *Application_Startup* происходит присвоение свойству *RootVisual* (Корневой визуальный элемент) приложения нового объекта *MainPage* (Главная страница). Тем самым объявляется, что UI объекта *Page* является первым экраном UI, который должно создать это приложение. Если

предполагается использовать другие экраны UI, объявленные в XAML, они будут запускаться уже из объекта *MainPage*.

Объект *MainPage* является XAML-объектом по умолчанию, создаваемым шаблоном для размещения UI вашего приложения.

Файлы MainPage.xaml и MainPage.xaml.cs

Файл MainPage.xaml является для приложения UI по умолчанию. При компиляции вместе с ассоциированным с ним файлом выделенного кода он формирует класс *MainPage*, экземпляр которого будет объектом *MainPage*. Если вспомнить из предыдущего раздела, в качестве значения свойства *RootVisual* приложения был задан новый объект *MainPage*, что позволило данному классу быть UI по умолчанию.

Вот XAML, формируемый по умолчанию для MainPage.xaml:

```
<UserControl x:Class="SlidingBlocks3.MainPage"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"

    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    Width="640"
    Height="480">

  <Grid x:Name="LayoutRoot" Background="White" >

  </Grid>
</UserControl>
```

Прежде всего, обратите внимание, что контейнером XAML является *UserControl*. Этого не было в Silverlight 1, не имеющем модели программирования .NET. Как говорилось ранее, при построении .NET-приложений Silverlight в Visual Studio, фактически, создаются элементы управления, которые компилируются в DLL и помещаются в XAP, который затем открывает и исполняет Silverlight.

В данном случае, видим, что экземпляр этого *UserControl* называется *SlidingBlocks3.MainPage*. *SlidingBlocks3* – это пространство имен (вернитесь к свойствам проекта), и *MainPage* – имя класса из этого пространства имен.

Объявления *xmlns* и *xmlns:x* задают пространство имен по умолчанию и дополнительное пространство имен, соответственно, которые должны использоваться для проверки корректности XAML. Ранее был рассмотрен атрибут *x:Class*, используемый для описания класса этого элемента управления, который также является примером применения дополнительного пространства имен, запись которого начинается с префикса *x:*.

Наконец, ширине и высоте присвоены значения по умолчанию: 640 × 480.

Далее идет корневой *Grid*. В Silverlight 3 корневым элементом должен быть *Container* (Контейнер), которым в данном случае является *Grid* под именем *LayoutRoot*. Соответственно, все элементы дизайна UI будут дочерними элементами этого узла.

Вот выделенный код для данного XAML:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Net;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
```

```
using System.Windows.Media.Animation;
using System.Windows.Shapes;

namespace SlidingBlocks3
{
   public partial class MainPage : UserControl
   {
      public MainPage()
      {
         // Необходима для инициализации переменных
         InitializeComponent();
      }
   }
}
```

Те, кто знаком с C#, видят, что этот код очень похож на код, который, возможно, вы использовали ранее. По сути, это стандартный класс *MainPage*, унаследованный от типа *UserControl*. В конструкторе класса выполняется общая настройка с помощью вызова метода *InitializeComponent()*. В разделе «Написание игры на Silverlight» данной главы, в котором мы займемся созданием головоломки с перемещением фрагментов, вы добавите в этот модуль характерный для вашей страницы код.

Веб-проект

Кроме проекта элемента управления, шаблон создает Веб-проект, в котором размещается ваше приложение Silverlight. Этот Веб-проект включает два ASPX-файла: Default.aspx, являющийся пустой Веб-формой, на базе которой можно построить свое приложение; и тестовую страницу <*ИмяПриложения*>TestPage.aspx (например, SlidingBlocks3TestPage.aspx), которая содержит все необходимое для запуска Silverlight из ASP.NET.

Хотя Silverlight не имеет никаких зависимостей на стороне сервера, ASP.NET предлагает элементы управления, обеспечивающие возможность создания JavaScript и HTML на стороне клиента, необходимых для размещения Silverlight в браузере.

Файл TestPage включает ссылки на эти элементы управления. Далее приведена разметка для ASPX-файла[1]:

```
<%@ Page Language="C#" AutoEventWireup="true" %>

<%@ Register Assembly="System.Web.Silverlight"
   Namespace="System.Web.UI.SilverlightControls" TagPrefix="asp" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml"style="height:100%;">
<head runat="server">
   <title>SlidingBlocks</title>
</head>
<body style="height:100%;margin:0;">
   <form id="form1" runat="server" style="height:100%;">

         <asp:ScriptManager ID="ScriptManager1" runat="server"></asp:ScriptManager>
      <div  style="height:100%;">
         <asp:Silverlight ID="Xaml1" runat="server"
                  Source="~/ClientBin/SlidingBlocks3.xap" MinimumVersion="3.0.40624.0"
                  Width="100%" Height="100%" />
      </div>
   </form>
</body>
</html>
```

1 Такой код генерировался в Visual Studio Tools for Silverlight 2, в Visual Studio Tools for Silverlight 3 используется обычный html-объект. По адресу http://go.microsoft.com/fwlink/?LinkId=153377 находится более детальная информация по этому вопросу (*прим. редактора*).

Заметьте, что данная технология является развивающейся, поэтому атрибуты номера версии и открытого ключа могут немного меняться. Если для создания кода вы использовали шаблон, не стоит волноваться, все будет в порядке.

На этой странице используются два элемента управления ASP.NET. Первый – *ScriptManager* (Диспетчер сценариев), который является артефактом AJAX ASP.NET и превосходным средством управления загрузкой и использованием всех необходимых библиотек JavaScript в соответствующее время и в соответствующем месте.

Второй – элемент управления *Silverlight*. Обратите внимание, что в качестве параметра он принимает рассмотренный выше XAP. Этот элемент управления будет формировать правильный HTML-код для создания <*object*>, представляющего Silverlight в браузере.

После запуска этой страницы будет сформировано много кода на HTML и JavaScript. Элемент управления Silverlight создается и получает файл XAP ближе к концу кода. Вот фрагмент кода:

```
<script type="text/javascript">
//<![CDATA[
Sys.Application.initialize();
Sys.Application.add_init(function() {
    $create(Sys.UI.Silverlight.Control,
        {"source":"ClientBin/SlidingBlocks3.xap"},
         null, null, $get("Xaml1_parent"));
        });
//]]>
</script>
```

Браузер интерпретирует этот сценарий и создает элемент управления *Silverlight*. Обратите внимание, что корректную работу элементов управления Silverlight ASP.NET обеспечит ASP.NET. Если экземпляр Silverlight создается не из ASP.NET, по-прежнему могут использоваться сценарии JavaScript, входящие в SDK. Это подробно рассматривается в Главе 6.

Написание игры на Silverlight

Silverlight поддерживает модель разделения дизайна и процесса разработки, при которой дизайн представляется с помощью XAML и код, обычно (хотя это не обязательно) на C#, создается в выделенном файле.

В данном примере XAML будет использоваться совсем немного, лишь для размещения *Canvas*, в котором будут находиться фрагменты рисунка и его полное изображение. Если вернуться к рис. 3-3, эти компоненты можно увидеть слева и справа, соответственно. Обратите внимание, что хотя главным контейнером страницы по умолчанию является *Grid*, для простоты в этой игре мы используем *Canvas*.

Создание UI в XAML

Как видно на рис. 3-3, UI этой игры очень прост. Он состоит из области экрана (занимаемой элементом *Canvas*), в которой располагаются перемещаемые фрагменты собираемого изображения, и области, в которой формируется визуальное представление собранного рисунка.

Вот пример XAML, который обеспечивает отображение *Canvas* и завершенного рисунка:

```
<UserControl x:Class="SlidingBlocks.MainPage"
          xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"

      xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
      Width="640"
      Height="480"
      >

   <Canvas x:Name="LayoutRoot" >
```

```
    <Canvas x:Name="GameContainer" />
    <Image Source="sl.jpg" Canvas.Left="500"
      Height="400" Width="400" Stretch="UniformToFill">
    </Image>
  </Canvas>
</UserControl>
```

Как видите, все очень просто. Canvas, в котором размещаются фрагменты собираемого изображения, называется *GameContainer* (Контейнер игры). Изображение, которое будет формировать визуальное представление собранного рисунка, является статическим, таким образом, не нуждается в имени. Изначально оно заполнено изображением sl.jpg из этого же Веб-проекта. Вот и весь дизайн. В следующем разделе займемся кодом.

Написание кода игры

Код игры в данном примере написан на C#. Его без труда можно было бы транслировать на VB.NET, IronPython или любой другой поддерживаемый язык программирования, но в большинстве случаев в данной книге используется C#.

Инициализация структур данных

Первый шаг в написании кода для этой игры – инициализация структур данных, которые будут использоваться приложением. В данном случае мы собираемся разбить рисунок на 16 фрагментов, 15 из которых будут выставлены на поле в произвольном порядке. Один фрагмент не используется, вместо него оставляем пустое пространство, обеспечивающее возможность перемещения фрагментов по игровому полю.

В Silverlight нельзя создать фрагмент существующего изображения, но можно вырезать его по определенному контуру. Не путайте вырезание с обрезкой. В первом случае вы указываете, какие части рисунка должны быть отрисованы; во втором – удаляете все, кроме необходимой части изображения. Silverlight не поддерживает обрезку, поэтому приходится вырезать изображение.

Проблема использования вырезания состоит в том, что при его применении остальная часть рисунка остается доступной и может быть активизирована щелчком мыши. Например, если имеется изображение 400 × 400 и из него вырезается квадратный фрагмент 100 × 100 в позиции (100,100), мы видим квадрат 100 × 100 в позиции (100,100), но объект 400 × 400 никуда не исчез: он стал пустым, но по-прежнему может быть активизирован щелчком мыши. Это не отвечает нашим требованиям для создания головоломки с перемещаемыми фрагментами. Но что можно сделать?

Решение состоит во введении в *Canvas* изображения, размеры которого соответствуют размерам вырезанного фрагмента, и последующем использовании трансформации транслирования на него, чтобы вырезанная часть изображения располагалась в верхнем левом углу данного *Canvas*, и, таким образом, *Canvas* формировал визуальное отображение только вырезанной области.

Итак, для головоломки 4 × 4, образованной 16 блоками необходимо создать 16 изображений и 16 объектов *Canvas*. Также потребуется как-то представлять поле, чтобы вы знали, какое изображение в каком квадрате находится. Вот код для описания этого:

```
Canvas[] cI = new Canvas[16];
Image[] i = new Image[16];
int[] board = new int[16];
```

Создание фрагментов головоломки

Чтобы создать фрагменты головоломки, понадобиться загрузить изображение sl.jpg в каждый элемент массива изображений. Для этого с помощью универсального идентификатора ресурса (uniform resource identifier, URI) мы указываем на необходимое изображение и определяем класс *BitmapImage*

(из пространства имен *System.Windows.Media.Imaging*), в который считываются данные по этому URI. И присваиваем этот класс элементу управления изображение:

```
i[nx].Source = new BitmapImage(uri);
```

Из конструктора можно вызвать функцию (*InitBoard*), которая будет использоваться для создания экземпляров фрагментов головоломки:

```
public MainPage()
{
   // Необходимо для инициализации переменных
   InitializeComponent();
   InitBoard();
}
```

Эту функцию можно увидеть здесь:

```
void InitBoard()
{
  Uri uri = new Uri("sl.jpg", UriKind.Relative);
  int nx = 0;
  for (int ix = 0; ix < 4; ix++)
    for (int iy = 0; iy < 4; iy++)
    {
      nx = (ix * 4) + iy;
      i[nx] = new Image();
      i[nx].Height = 400;
      i[nx].Width = 400;
      i[nx].Stretch = Stretch.UniformToFill;
      RectangleGeometry r = new RectangleGeometry();
      r.Rect = new Rect((ix * 100), (iy * 100), 100, 100);
      i[nx].Clip = r;
      i[nx].Source = new BitmapImage(uri);
      i[nx].SetValue(Canvas.TopProperty, Convert.ToDouble(iy * 100 * -1));
      i[nx].SetValue(Canvas.LeftProperty, Convert.ToDouble(ix * 100 * -1));

      cI[nx] = new Canvas();
      cI[nx].Width = 100;
      cI[nx].Height = 100;
      cI[nx].Children.Add(i[nx]);
      cI[nx].SetValue(Canvas.NameProperty, "C" + nx.ToString());
      cI[nx].MouseLeftButtonDown += new
        MouseButtonEventHandler(Page_MouseLeftButtonDown);
      if (nx < 15)
        GameContainer.Children.Add(cI[nx]);
    }

  // Перемешиваем фрагменты
  shuffle();

  // Отрисовываем поле
  drawBoard();

}
```

Не забудьте добавить ссылку на *System.Windows.Media.Imaging* в начале кода:

```
using System.Windows.Media.Imaging;
```

На первый взгляд все это может показаться немного сложным, но при более близком рассмотрении все предельно просто. Задается вложенный цикл для прохождения по осям x и y с диапазоном значений от 0 до 3 . Это, как можно догадаться, используется для организации массива 4 × 4 для изображений.

Массивы *Image* и *Canvas*, используемые для хранения фрагментов, являются одномерными массивами по 16 элементов каждый (при такой структуре для их хранения требуется меньше памяти и кода). Чтобы спроецировать двумерные координаты *ix, iy* на одномерный массив, необходимо выполнить следующие вычисления:

```
nx = (ix * 4) + iy;
```

Для каждого элемента задаются размеры 400 × 400 с помощью значения *UniformToFill* (Равномерное заполнение) свойства Stretch (Растяжение). Более подробно об использовании изображений рассказывается в Главе 4, «Основы XAML».

Далее вычисляется область вырезания. Для этого используется объект *RectangleGeometry* (Геометрический элемент прямоугольник):

```
RectangleGeometry r = new RectangleGeometry();
r.Rect=new Rect((ix*100), (iy*100) ,100,100);
i[nx].Clip = r;
```

Таким образом, определяется прямоугольник с соответствующими координатами (полученными путем умножения *ix* и *iy* на 100) соответствующего размера (100 × 100), и он задается как область вырезания для текущего изображения (*i[nx]*).

Далее изображение загружается в *BitmapImage*, инициализация которого происходит путем задания URI этого изображения, и помещается в заданное положение. Чтобы изображение, вырезанное в позиции (100,100), появилось в верхнем левом углу, оно должно быть передвинуто на (−100,−100) относительно вырезанной области. Таким образом, свойствам *Top* (Сверху) и *Left* (Слева) должно быть задано значение −100 и умножено на значения *iy* и *ix*, соответственно.

```
i[nx].Source = new BitmapImage(uri);
i[nx].SetValue(Canvas.TopProperty, Convert.ToDouble(iy * 100 * -1));
i[nx].SetValue(Canvas.LeftProperty, Convert.ToDouble(ix * 100 * -1));
```

Изображения до сих пор не добавлены в родительский *Canvas*. Это следующий шаг. *Canvas* необходимо инициализировать, задать его размеры и добавить в него в качестве дочернего элемента соответствующее изображение. Вот код:

```
cI[nx] = new Canvas();
cI[nx].Width = 100;
cI[nx].Height = 100;
cI[nx].Children.Add(i[nx]);
```

Теперь вы получили свой *Canvas*, и он содержит вырезанное изображение. Завершаем инициализацию, присваивая *Canvas* имя (чтобы его можно было отслеживать, когда вы щелкаете его), определяем обработчик событий для описания действий, которые будут выполняться по щелчку пользователя по нему, и, наконец, добавляем его в родительский *Canvas*. Пока нет необходимости задавать местоположение фрагмента. Это будет сделано после того, как фрагменты будут перемешаны на поле. Обратите внимание, последнее изображение не добавляется на поле, потому что мы хотим, чтобы осталось свободное пространство.

```
cI[nx].SetValue(Canvas.NameProperty, "C" + nx.ToString());
cI[nx].MouseLeftButtonDown += new
   MouseButtonEventHandler(Page_MouseLeftButtonDown);
if(nx<15)
   GameContainer.Children.Add(cI[nx]);
```

Наконец, необходимо перемешать фрагменты и отрисовать игровое поле. Код для реализации этого представлен в следующем разделе.

Перемешиваем фрагменты

Фрагменты головоломки перемешиваются с помощью довольно простого алгоритма перемешивания, который перебирает массив 100 раз, выбирая два случайных элемента каждый раз; если эти два элемента разные, он меняет местами их содержимое. В конце он загружает в последний элемент значение -1, чтобы обозначить пустой квадрат.

Здесь представлен алгоритм перемешивания:

```
void shuffle()
{
 // Инициализируем игровое поле
 for (int n = 0; n < 15; n++)
 {
  board[n] = n;
 }
 Random rand = new Random(System.DateTime.Now.Second);
 for (int n = 0; n < 100; n++)
 {
  int n1 = rand.Next(15);
  int n2 = rand.Next(15);
  if (n1 != n2)
  {
   int tmp = board[n1];
   board[n1] = board[n2];
   board[n2] = tmp;
  }
 }
 board[15] = -1;

}
```

Следующий шаг после того, как фрагменты перемешаны, – отрисовка игрового поля.

Отрисовка игрового поля

На данный момент мы имеем все фрагменты головоломки, определенные как элементы *Image* в элементе *Canvas*, и массив целых чисел, значение индекса n которого соответствует фрагменту, отображаемому в данном месте. Вы также перемешали этот массив целых чисел, поэтому пришло время отрисовать игровое поле. Это делается следующим образом:

```
void drawBoard()
{
 int nx = 0;
 int ny = 0;
 for (int n = 0; n < 15; n++)
 {
  nx = n / 4;
  ny = n % 4;
  if(board[n]>=0)
  {
   cI[board[n]].SetValue(Canvas.TopProperty, Convert.ToDouble(ny * 100));
   cI[board[n]].SetValue(Canvas.LeftProperty, Convert.ToDouble(nx * 100));

  }
 }
}
```

В данном случае выполняется цикл с изменением индекса от 0 до 14 (в головоломке 15 фрагментов) и вычисление координат *x,y* для каждого блока в сетке 4 × 4. Значение *x* – это просто результат целочисленного деления индекса массива на 4, и значение *y* – это модуль индекса, деленного на 4. Умножаем эти значения на 100 и получаем необходимые координаты для отрисовки элемента *Canvas*.

Теперь мы имеем полностью инициализированную игру. Рисунок располагается справа, а игровое поле с перемешанными фрагментами рисунка – слева.

Обработка действий пользователя

Теперь необходимо заняться обработкой действий пользователя. В играх подобного рода предполагается, что пользователь будет щелкать фрагмент изображения, и если этот фрагмент находится рядом с пустой областью, он плавно переместится в эту область, оставляя пустым занимаемое до этого место. Итак, требуется обработать щечки по элементу *Canvas*, содержащего фрагмент изображения. Если вернуться к инициализации фрагментов, можно вспомнить такую строку:

```
cI[nx].MouseLeftButtonDown
    += new MouseButtonEventHandler(Page_MouseLeftButtonDown);
```

Здесь описывается, что по щелчку *Canvas* будет запускаться обработчик события *Page_MouseLeftButtonDown*. Этот обработчик события задан для каждого из блоков *Canvas*.

Код этого обработчика события состоит из двух разделов. Первый определяет, который из *Canvas* сформировал событие и положение этого *Canvas* на игровом поле:

```
void Page_MouseLeftButtonDown(object sender, MouseButtonEventArgs e)
{
 Canvas c = sender as Canvas;
 int nCanvasID = -1;
 int nBoardLoc = -1;
 int nEmptyLoc = -1;
 for (int i = 0; i < 16; i++)
 {
  if (c == cI[i])
  {
   nCanvasID = i;
   break;
  }
 }
 for (int i = 0; i < 16; i++)
 {
  if (board[i] == nCanvasID)
  {
   nBoardLoc = i;
  }
  else if (board[i] == -1)
  {
   nEmptyLoc = i;
  }
 }
```

Итак, например, игрок щелкнул фрагмент, представляющий верхний левый угол полного изображения (блок 0), но в настоящий момент он находится в нижнем левом углу игрового поля (позиция 12). При возникновении события щелчка, выполняется перебор всего массива элементов *Canvas*, представляющих фрагменты головоломки, до тех пор, пока не будет найден фрагмент, соответствующий *Canvas*, по которому было произведен щелчок. И здесь вы можете получить индекс этого элемента в массиве, сохранив его в переменной *nCanvasID* (в предыдущем гипотетическом примере он равнялся бы 0). После этого, просканировав игровое поле, можно определить местоположение элемента 0 на нем и присвоить значение его позиции переменной *nBoardLoc* (которое в гипотетическом примере равно 12). И, находясь в этой позиции, найдите местоположение пустой ячейки на игровом поле и сохраните его в *nEmptyLoc*.

Вторая часть кода проверяет возможность перемещения фрагмента, и, если перемещение возможно, перемещает фрагмент в пустую ячейку и соответствующим образом обновляет игровое поле.

```
// Прверяем возможность перемещения
if ((nBoardLoc == nEmptyLoc + 1) ||
   (nBoardLoc == nEmptyLoc - 1) ||
   (nBoardLoc == nEmptyLoc + 4) ||
   (nBoardLoc == nEmptyLoc - 4))
{
   int nx = nEmptyLoc/4;
   int ny = nEmptyLoc%4;


  cI[board[n]].SetValue(Canvas.TopProperty, Convert.ToDouble(ny * 100));
  cI[board[n]].SetValue(Canvas.LeftProperty, Convert.ToDouble(nx * 100));
   board[nEmptyLoc] = nCanvasID;
   board[nBoardLoc] = -1;

   checkWinner();
}
```

```
            else
            {
               // не делаем ничего
            }


         }
```

Для этого сначала проверим местоположение пустой ячейки относительно положения блока, по которому был произведен щелчок. Если они располагаются непосредственно рядом друг с другом (над, под, слева или справа), перемещение возможно. Поскольку игровое поле 4 × 4 является одномерным массивом, сделать это несложно. Позиции элементов, располагающихся слева и справа от текущего элемента, отличаются от текущей на -1 и +1, соответственно, и позиции элементов сверху и снизу – на -4 и +4, соответственно. Так что если результат проверки фрагментов указывает на пустую ячейку, игрок знает, что можете двигаться.

Чтобы выполнить перемещение, необходимо получить координаты *x* и *y* относительно положения пустого фрагмента и задать их как координаты блока, по которому был произведен щелчок. Затем присвоить позиции поля, на которой до этого находилась пустая ячейка, значение *Canvas*, по которому был произведен щелчок, и позиции поля, на которой до этого располагался *Canvas*, по которому был произведен щелчок, значение -1, указывая на то, что теперь эта ячейка пуста.

Наконец, вызовем функцию *checkWinner()*, чтобы проверить успешность решения головоломки.

Проверка условия выигрыша

Функция *checkWinner* (Проверка победителя) проверяет успешность решения головоломки. Опять же, все очень просто. Головоломка решена, если каждый *n*-ный элемент игрового поля равен *n*, т.е. *Canvas* 0 имеет индекс 0, *Canvas* 1- индекс 1 и т.д.

Самый эффективный способ сделать это – предположить, что головоломка решена успешно, и просканировать игровое поле в поисках несоответствия *board[n]* значению *n*, что будет свидетельствовать о неверном решении и приводить к завершению цикла. Если все индексы совпадают, у нас есть победитель!

```
void checkWinner()
      {
         bool bCompleted = true;
         for (int n = 0; n < 15; n++)
         {
            if (n != board[n])
            {
               bCompleted = false;
               break;
            }
         }
         if (bCompleted)
         {
            // Игрок выиграл....сделайте для него что-нибудь приятное.
         }
      }
```

Теперь вы получили все элементы простой игры-головоломки с перемещением фрагментов, написанной на C# для Silverlight. И для этого необходимо всего лишь 100 строк кода. По ходу чтения этой книги данный пример можно расширять, добавляя анимацию перемещения блоков, сохранение лучших результатов, возможность загрузки изображений в приложение и т.д. Нет пределов совершенству!

Заключение

В данной главе мы рассмотрели Visual Studio 2008 и предлагаемые ею различные инструменты и шаблоны для разработки приложений Silverlight с использованием языков программирования .NET. Был детально разобран проект, созданный на базе стандартных шаблонов Silverlight, рассмотрены все его файлы и то, как он используется для разработки и развертывания приложения Silverlight. После этого мы применили теорию на практике и создали полнофункциональную игру-головоломку с перемещением фрагментов, используя XAML и C#.

Это позволило почувствовать возможности Silverlight, но мы лишь слегка прикоснулись к ним. В Части 2 данной книги, «Программирование на Silverlight 3 с использованием .NET», будут более глубоко рассмотрены функциональные возможности, доступные в .NET Framework, включая создание собственных элементов управления, работу с сетью и обмен информацией, данные и XML, динамические языки программирования и серверные элементы управления ASP.NET.

Глава 4

Основы XAML Silverlight

Расширяемый язык разметки приложений (Extensible Application Markup Language, XAML) – это ядро приложения Silverlight. Он используется для определения графических ресурсов, взаимодействий, анимаций и временных шкал. XAML основывается на Расширяемом языке разметки (Extensible Markup Language, XML), поэтому все описывается в текстовом формате с использованием атрибутов для объявления свойств, методов и событий.

В данной главе будут рассмотрены общие компоненты XAML, используемые для описания визуальных элементов приложения. Во-первых, мы остановимся на формировании компоновки, в том числе на том, как элементы могут размещаться на экране относительно их контейнеров и друг друга. Также рассмотрим разные кисти, используемые для заливки фигур и элементов управления, а также обводки для отрисовки контуров. Затем вы узнаете о контурах и геометрических элементах и о том, как они используются для создания сложных фигур и групп фигур, а также как с их помощью реализовывается вырезание других объектов или заливки. Наконец, будут представлены элементы управления, использующие эти компоненты XAML, включая *Canvas*, который является основным элементом компоновки Silverlight. Кроме того, мы познакомимся, как использовать элементы управления *Glyphs* (Глифы) и *TextBlock* для отображения текста.

Компоновочные свойства XAML

Если говорить кратко, свойства *Canvas.Left* и *Canvas.Top* используются для компоновки элементов управления в XAML. Их называют *присоединенными* свойствами, что всего лишь означает, что они глобально доступны в коде XAML или изменяют свойство, определенное родительским элементом. Кроме того, присоединенное свойство *Canvas.ZIndex* может использоваться для обозначения положения объекта на плоскости относительно оси Z, что определяет очередность отрисовки в случае наложения объектов друг на друга. По умолчанию элемент, отрисовываемый последним (самый нижний в документе XAML), является самым верхним, но с помощью *Canvas.ZIndex* это положение можно переопределить.

Итак, рассмотрим следующий фрагмент XAML. Он представляет код для *Canvas* с двумя прямоугольниками. Для определения положения верхнего левого угла этих прямоугольников относительно холста, в котором они располагаются, используются свойства *Canvas.Left* и *Canvas.Top*.

```
<Canvas>
  <Rectangle Fill="Red" Width="200" Height="128"
         Canvas.Left="8" Canvas.Top="8"/>
  <Rectangle Fill="Black" Width="280" Height="80"
        Canvas.Left="40" Canvas.Top="32"/>
</Canvas>
```

Черный прямоугольник отрисовывается поверх красного, как показано на рис. 4-1.

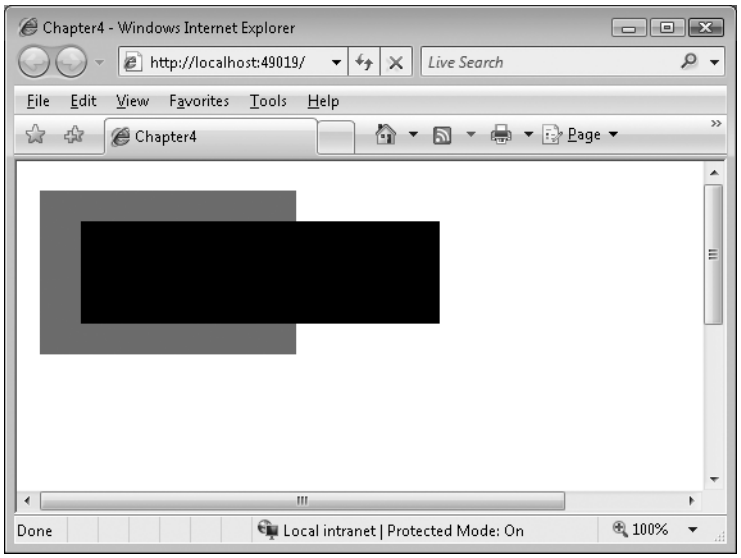

РИС. 4-1 Компоновка прямоугольников.

Красный прямоугольник находится на расстоянии 8 пикселов от левого края и 8 пикселов от верхней границы родительского холста. Черный сдвинут на 40 пикселов от левого и 32 пиксела от верхнего края. Черный прямоугольник располагается поверх красного, так как он визуализировался последним.

Это положение элементов можно переопределить с помощью свойства *ZIndex*. *ZIndex* принимает числовые значения. Объекты с большими значениями *ZIndex* будут отображаться поверх объектов с меньшими *ZIndex*, как видно в следующем коде:

```
<Canvas>
  <Rectangle Canvas.ZIndex="2"
         Fill="Red" Width="200" Height="128"
         Canvas.Left="8" Canvas.Top="8"/>
  <Rectangle Canvas.ZIndex="1"
         Fill="Black" Width="280" Height="80"
         Canvas.Left="40" Canvas.Top="32"/>
</Canvas>
```

Как показано на рис. 4-2, теперь красный прямоугольник располагается поверх черного, чего и следовало ожидать, поскольку значение свойства *ZIndex* красного прямоугольника больше, чем значение этого свойства для черного прямоугольника.

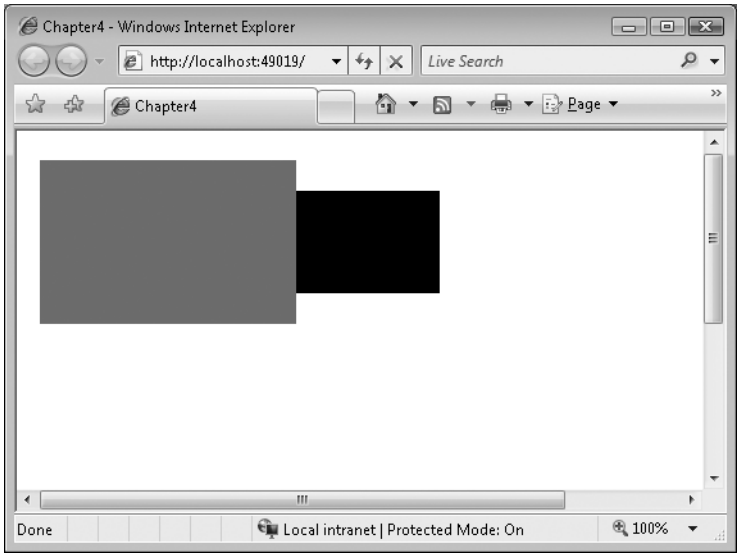

РИС. 4-2 Изменение порядка расположения элементов относительно оси Z.

Кисти XAML

Кисти в XAML определяют, как отрисовываются и закрашиваются фигуры. В предыдущем примере представлено два прямоугольника, заливка которых выполнена с использованием известных цветов, *Red* (красный) и *Black* (черный). Это простые примеры применения кистей. В следующих разделах будут описаны более сложные типы *кистей*, поддерживаемые XAML.

SolidColorBrush

SolidColorBrush заполняет область сплошным цветом. Для задания цвета может использоваться именованное значение, например *Red* или *Black*, или шестнадцатеричные значения, обозначающие интенсивности альфа, красного, зеленого и синего каналов. Например, белый цвет в шестнадцатеричном представлении описывается как *#FFFFFFFF*, и красный – *#FFFF0000*.

LinearGradientBrush

LinearGradientBrush (Линейная градиентная кисть) заполняет область линейной градиентной заливкой, определенной в двумерном пространстве. Градиент по умолчанию описывается с использованием нормированного прямоугольника (прямоугольника, верхний левый угол которого имеет координаты (0,0), и нижний правый угол – координаты (1,1)). В этом случае градиент распространяется в направлении от верхнего левого угла к нижнему правому углу. Если для каждой из этих точек задать цвет, Silverlight отобразит плавный переход между ними.

В качестве примера рассмотрим следующее описание прямоугольника:

```
<Canvas>
  <Rectangle Width="200" Height="128" Canvas.Left="8" Canvas.Top="8">
    <Rectangle.Fill>
      <LinearGradientBrush>
        <GradientStop Color="#FF000000" Offset="0"/>
        <GradientStop Color="#FFFFFFFF" Offset="1"/>
      </LinearGradientBrush>
    </Rectangle.Fill>
  </Rectangle>
</Canvas>
```

Этот фрагмент XAML-кода определяет *LinearGradientBrush*, распространяющуюся из верхнего левого угла в нижний правый угол прямоугольника. Первая точка градиента, в начале градиента, черная ((#FF000000), и вторая точка градиента, в конце градиента, белая ((#FFFFFFFF). Прорисовку этого прямоугольника можно увидеть на рис. 4-3.

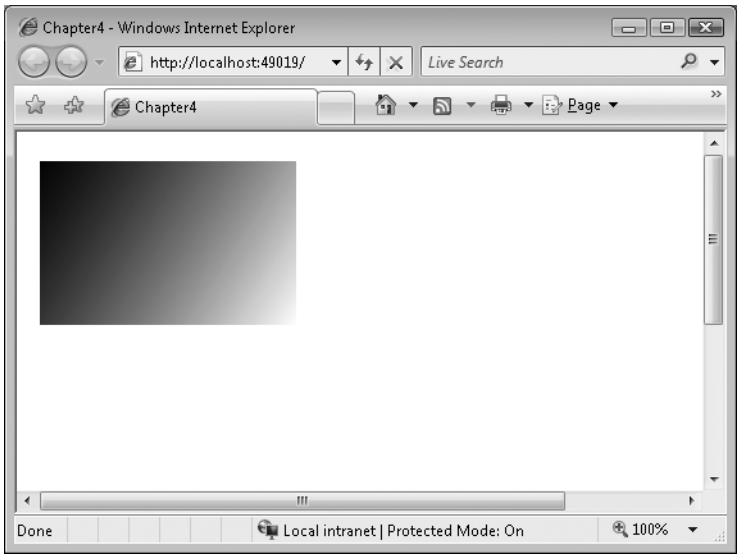

РИС. 4-3 Использование *LinearGradientBrush*.

Изменение направления распространения градиента

Задавая свойства *StartPoint* (Начальная точка) и *EndPoint* (Конечная точка) *LinearGradientBrush*, можно изменять направление распространения градиента. Чтобы направить градиентную заливку из нижнего левого в верхний правый угол, этим свойствам необходимо задать значения (0,1) и (1,0), соответственно:

```
<Rectangle Width="200" Height="128" Canvas.Left="8" Canvas.Top="8">
  <Rectangle.Fill>
    <LinearGradientBrush StartPoint="0,1" EndPoint="1,0">
      <GradientStop Color="#FF000000" Offset="0"/>
      <GradientStop Color="#FFFFFFFF" Offset="1"/>
    </LinearGradientBrush>
  </Rectangle.Fill>
</Rectangle>
```

На рис. 4-4 показано, как этот прямоугольник будет выглядеть на экране.

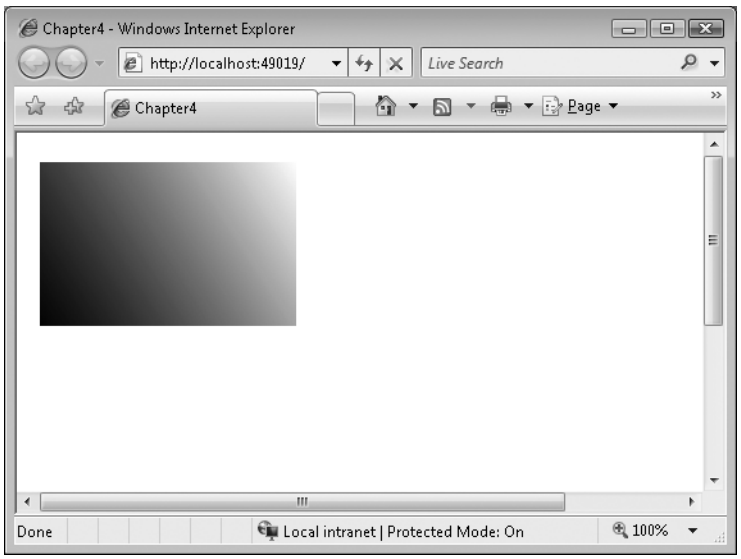

РИС. 4-4 Изменение направления градиентной заливки.

Добавление точек градиента

В предыдущем примере показано минимально возможное количество точек градиента, которое равно двум. Для управления градиентом можно задавать дополнительные точки градиента и цвета для них.

Например, если требуется получить градиентную заливку с изменением цвета от черного к белому и назад к черному, можно определить три точки следующим образом:

```
<Rectangle Width="200" Height="128" Canvas.Left="8" Canvas.Top="8">
  <Rectangle.Fill>
    <LinearGradientBrush>
      <GradientStop Color="#FF000000" Offset="0"/>
      <GradientStop Color="#FFFFFFFF" Offset="0.5"/>
      <GradientStop Color="#FF000000" Offset="1"/>
    </LinearGradientBrush>
  </Rectangle.Fill>
</Rectangle>
```

Первая точка, в позиции 0, черная; вторая, на полпути вдоль линии градиента (в позиции 0,5), белая; и третья точка, в конце градиента в позиции 1, снова черная. Если сформировать визуальное представление этого, можно будет увидеть примерно такой прямоугольник, как представлен на рис. 4-5.

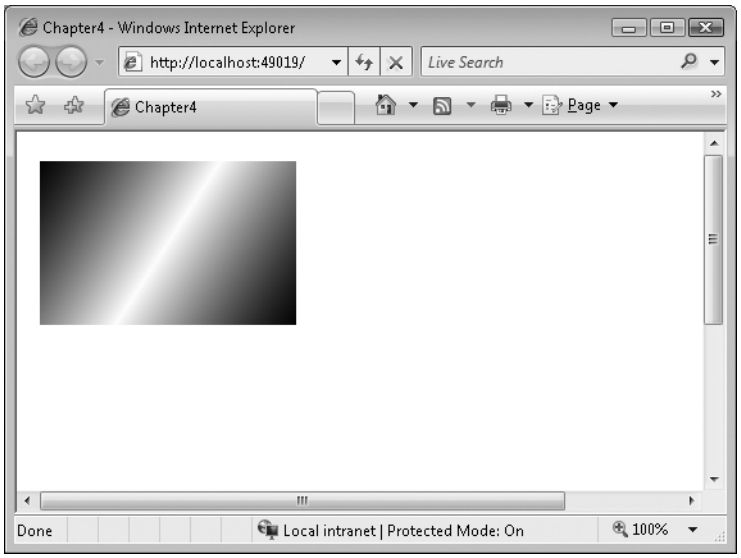

РИС. 4-5 Использование точек градиента.

Позиционирование точек градиента выполняется с помощью параметра *Offset* (Смещение), поэтому чтобы передвинуть белую секцию данного градиента ближе к верхнему левому углу, необходимо просто изменить значение его свойства *Offset* так, чтобы оно было ближе к нулю, как показано в данном фрагменте XAML:

```
<Rectangle Width="200" Height="128" Canvas.Left="8" Canvas.Top="8">
  <Rectangle.Fill>
    <LinearGradientBrush>
      <GradientStop Color="#FF000000" Offset="0"/>
      <GradientStop Color="#FFFFFFFF" Offset="0.1"/>
      <GradientStop Color="#FF000000" Offset="1"/>
    </LinearGradientBrush>
  </Rectangle.Fill>
</Rectangle>
```

То, как это меняет вид прямоугольника, можно увидеть на рис. 4-6.

Экспериментируя с позиционированием и направлениями градиента, можно добиться некоторых интересных эффектов. Эти свойства могут использоваться для заполнения фигур или определения контуров, как будет показано далее в данной главе.

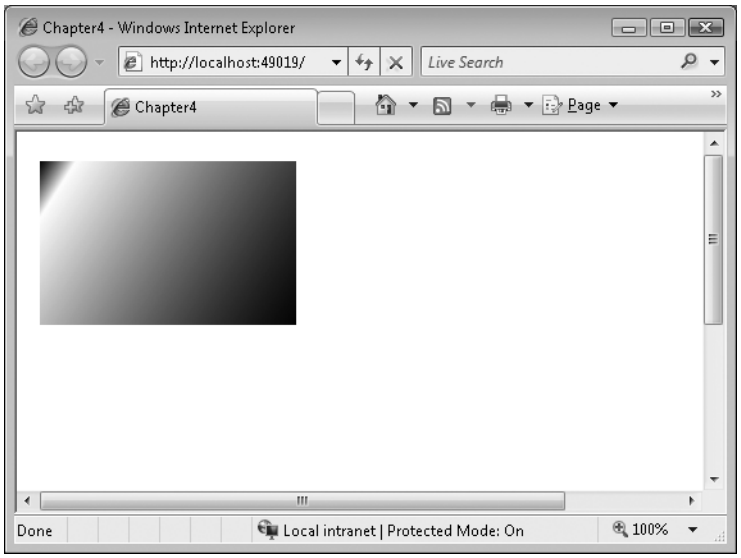

РИС. 4-6 Позиционирование точки градиента.

RadialGradientBrush

RadialGradientBrush (Радиальная градиентная кисть) аналогична *LinearGradientBrush* с точки зрения определения, но она задает круговой градиент, где точка 0 обозначает центр круга, и 1 обозначает внешний край. Проще показать это на примере, рассмотрев такой XAML:

```
<Rectangle Width="200" Height="128" Canvas.Left="8" Canvas.Top="8">
  <Rectangle.Fill>
    <RadialGradientBrush>
      <GradientStop Color="#FF000000" Offset="0"/>
      <GradientStop Color="#FFFFFFFF" Offset="1"/>
    </RadialGradientBrush>
  </Rectangle.Fill>
</Rectangle>
```

Этот код обеспечивает заполнение фигуры градиентной заливкой, начиная от черного цвета в центре и заканчивая белым по краям, как показано на рис. 4-7. Обратите внимание, что вы видите эллипс, так как внешний край прямоугольника белый, и фоновый цвет прямоугольника аналогичен цвету внешней точки градиента.

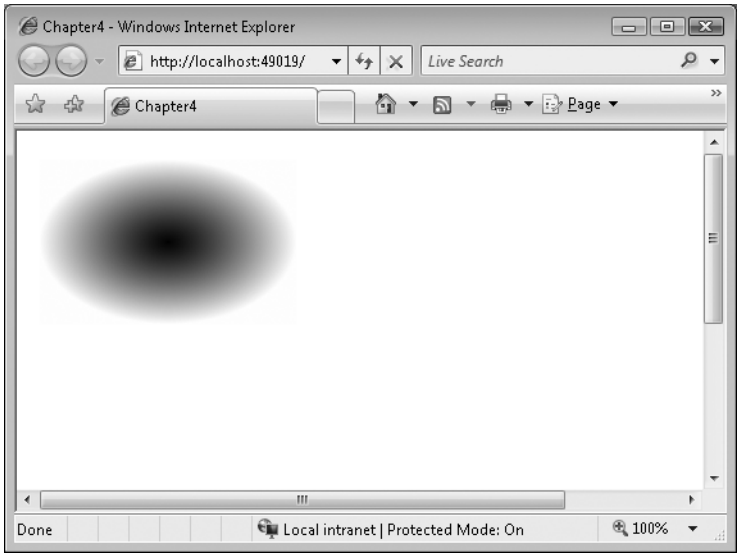

РИС. 4-7 Заполнение прямоугольника с помощью *RadialGradientBrush*.

Точки градиента для *RadialGradientBrush* определяются теми же методами, что и для *LinearGradientBrush*.

Задание фокуса

При выполнении заливки с использованием *RadialGradientBrush,* установив свойство *GradientOrigin* (Центр градиента), можно задавать фокус заливки. Это свойство используется для задания точки начала градиента, которая обычно располагается в центре круга. Несмотря на радиальную природу *RadialGradientBrush*, фокус задается в прямоугольном нормированном пространстве. Поэтому, если необходимо расположить фокус в верхнем левом углу, задайте *GradientOrigin* значение (0,0); если желаете расположить его в нижнем правом углу, задайте *GradientOrigin* значение (1,1). Следующий пример показывает градиентную заливку, фокусная точка которой располагается в нижнем правом углу объекта с координатами (0.7,0.7):

```
<Rectangle Width="200" Height="128" Canvas.Left="8" Canvas.Top="8">
  <Rectangle.Fill>
    <RadialGradientBrush GradientOrigin="0.7, 0.7">
      <GradientStop Color="#FF000000" Offset="0"/>
      <GradientStop Color="#FFFFFFFF" Offset="1"/>
    </RadialGradientBrush>
  </Rectangle.Fill>
</Rectangle>
```

На РИС. 4-8 представлено, как это выглядит.

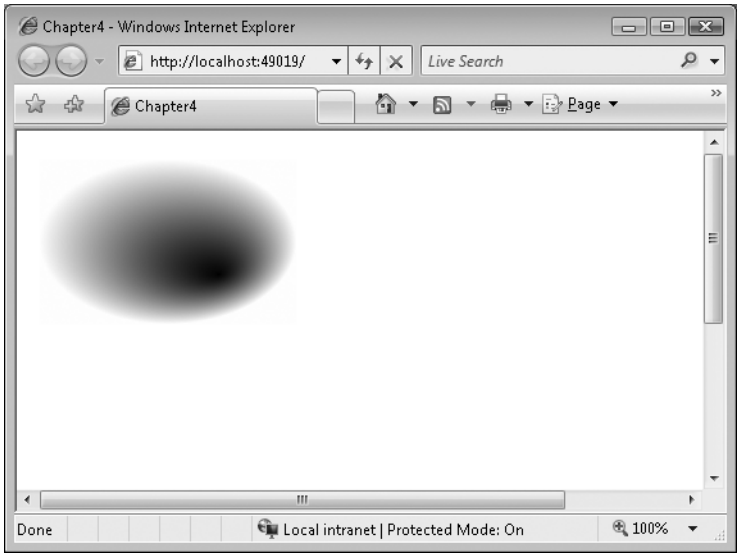

РИС. 4-8 Задание фокуса *RadialGradientBrush*.

Изменение SpreadMethod

С помощью свойства *SpreadMethod* (Метод распространения) можно определять характер повторения градиента. Существует три возможных значения *SpreadMethod*: *Pad* (Заполнение), *Reflect* (Отражение) и *Repeat* (Повтор). *Pad* заполняет круг градиентной заливкой соответственно заданным параметрам и является значением по умолчанию. На рис. 4-7 и 4-8 показана прорисовка базовой *RadialGradientBrush*, для которой задан метод распространения *Pad*.

Ниже представлен XAML, описывающий прямоугольник, заполненный градиентной заливкой, для свойства *SpreadMethod* которой задано значение *Reflect*. Результаты можно увидеть на рис. 4-9.

```
<Rectangle Width="200" Height="128" Canvas.Left="8" Canvas.Top="8">
  <Rectangle.Fill>
    <RadialGradientBrush SpreadMethod="Reflect">
      <GradientStop Color="#FF000000" Offset="0"/>
      <GradientStop Color="#FFFFFFFF" Offset="1"/>
    </RadialGradientBrush>
  </Rectangle.Fill>
</Rectangle>
```

Это приводит к отражению градиента, как видно на рис. 4-9. Для данного градиента определено, что он должен изменяться от черного к белому, но он возвращается опять к черному, как будто отражается. Аналогично, можно использовать метод *Repeat* для повторения градиента от черного к белому. На рис. 4-10 показан результат этого. Шаблон градиента повторяется с того места, где при обычных условиях он был бы завершен, повторяя переход от черного к белому вплоть до внешних краев прямоугольника.

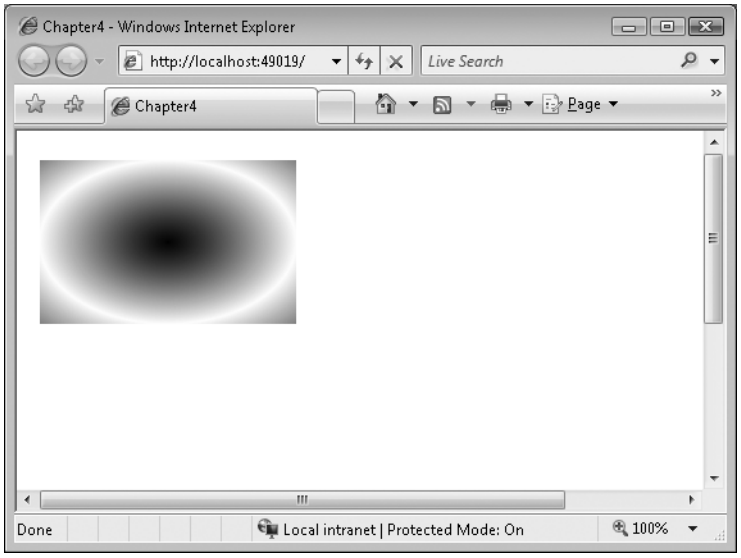

РИС. 4-9 Использование *SpreadMethod* со значением *Reflect*.

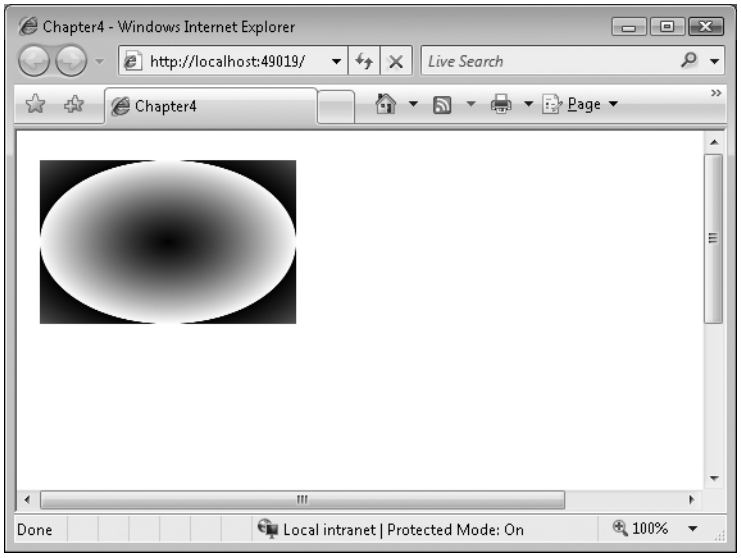

РИС. 4-10 Использование *SpreadMethod* со значением *Repeat*.

Задание радиуса *RadialGradientBrush*

Для задания необходимого радиуса градиентной заливки используются свойства *RadiusX* и *RadiusY*. Значение по умолчанию для этих свойств – 0,5. Если задать меньшее значение, будет закрашено больше одного круга с учетом поведения прорисовки, которое определяется методом *SpreadMethod*. Если задать значение, больше 0,5, градиент будет «растянут».

Например, следующий фрагмент XAML-кода определяет *RadialGradientBrush*, для свойств *RadiusX* и*RadiusY* которой задано значение 0,1, и свойство *SpreadMethod* не задано (т.е. по умолчанию оно принимает значение *Pad*).

```
<Rectangle Width="200" Height="128" Canvas.Left="8" Canvas.Top="8">
  <Rectangle.Fill>
    <RadialGradientBrush RadiusX="0.1" RadiusY="0.1">
```

```
      <GradientStop Color="#FF000000" Offset="0"/>
      <GradientStop Color="#FFFFFFFF" Offset="1"/>
    </RadialGradientBrush>
  </Rectangle.Fill>
</Rectangle>
```

Этот код обеспечит формирование визуального представления прямоугольника, заливка которого выполняется с помощью *RadialGradientBrush* с радиусом 0,1, поэтому она в пять раз меньше объектов, которые были представлены ранее. Это можно увидеть на рис. 4-11.

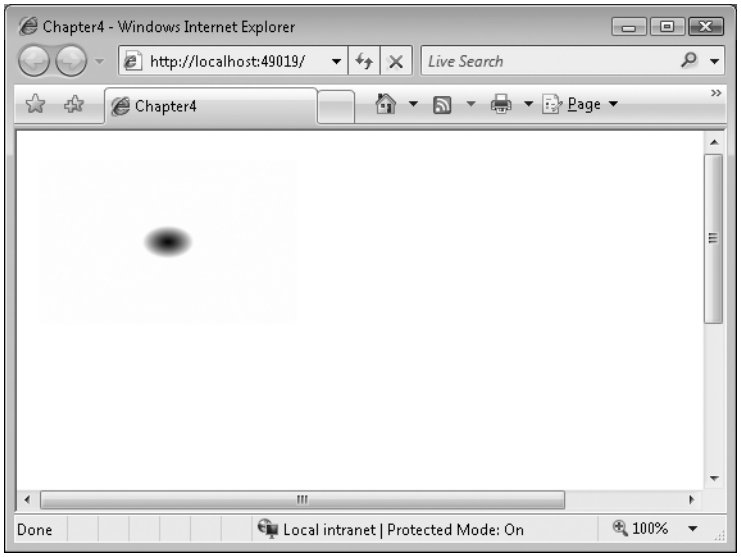

РИС. 4-11 Задание радиуса *RadialGradientBrush*.

В сочетании с методом *SpreadMethod* можно получить некоторые интересные эффекты. Пример задания значения *Reflect* для *SpreadMethod* представлен на рис. 4-12.

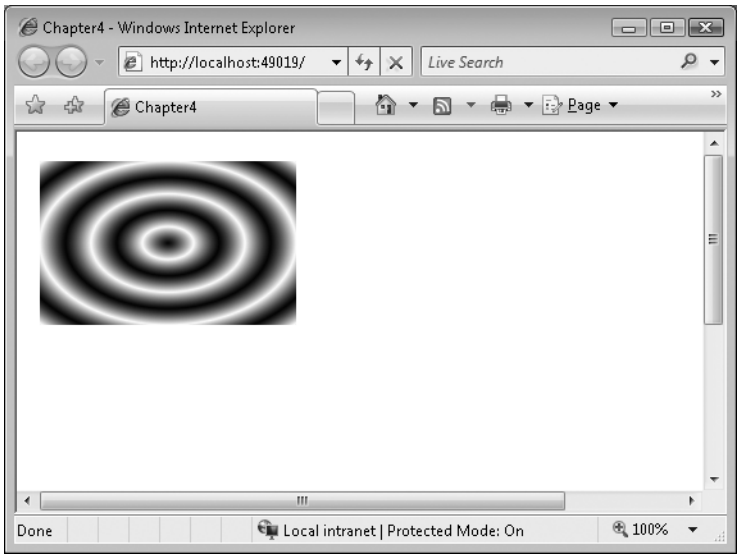

РИС. 4-12 Изменение значений радиуса в сочетании с изменением значений *SpreadMethod*.

Использование *ImageBrush*

Для заполнения области изображением используется *ImageBrush* (Кисть-изображение). Согласно поведению, по умолчанию кисть меняет размер соответственно размеру изображения, обеспечивая сохранение его пропорций. Следующий XAML-код обеспечивает заполнение содержимого прямоугольника *ImageBrush*:

```
<Rectangle Width="200" Height="128" Canvas.Left="8" Canvas.Top="8">
  <Rectangle.Fill>
    <ImageBrush ImageSource="smily.jpg" />
```

```
  </Rectangle.Fill>
</Rectangle>
```

Результаты представлены на рис. 4-13.

Растягивание изображения

С помощью свойства *Stretch* (Растяжение) можно определить то, как изображение заполняет закрашиваемую область. Это можно задать с помощью ряда различных режимов растяжения: *None* (Нет), *Uniform* (Равномерное), *UniformToFill* (Равномерное заполнение) и *Fill* (Заполнение).

Режим *None* формирует визуальное представление изображения без изменений, в данном случае растяжение не применяется. Значение *Uniform* обусловливает изменение размеров изображения так, чтобы оно заполняло прямоугольник, но при этом не меняет его пропорции. *UniformToFill* изменяет размер изображения так, чтобы оно максимально заполняло область вывода, но при этом не меняет его пропорции (обрезая изображение в случае необходимости). Значение *Fill* обеспечит заполнение изображением всей области вывода с использование независимого масштабирования по осям x и y. Таким образом, изображение будет искажено, но полностью заполнит доступную область.

РИС. 4-13 Заливка прямоугольника с помощью *ImageBrush*.

Все эти варианты в действии можно увидеть на рис. 4-14, на котором представлено четыре прямоугольника, заливка которых выполнена одним и тем же изображением, но с разными режимами растяжения.

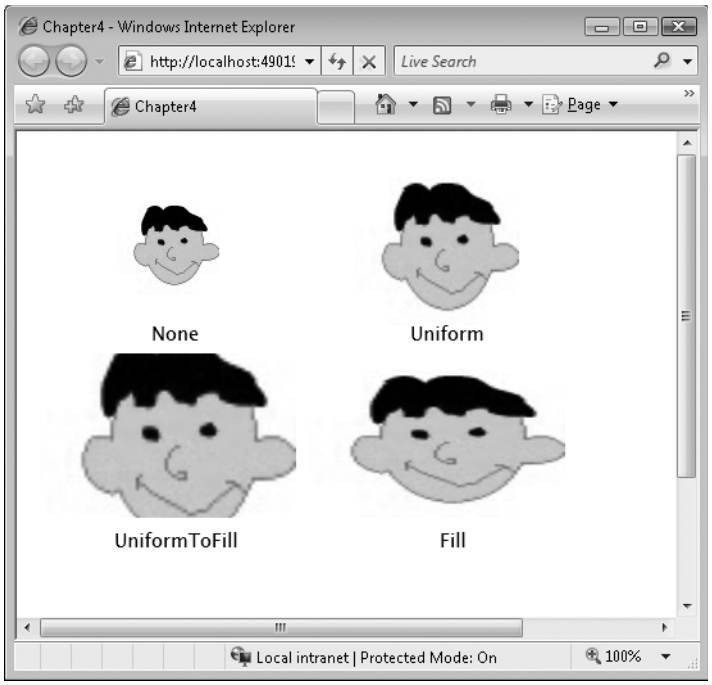

РИС. 4-14 Использование разных режимов свойства *Stretch* в *ImageBrush*.

Выравнивание изображения

Выравнивание изображения по осям x и y определяется с помощью свойств *AlignmentX* и *AlignmentY*. Изображение может быть выровнено вдоль оси x по левому краю, по правому краю, по центру и вдоль оси y по верху, центру или низу. Обратите внимание, что если изображение растягивается для заполнения области, задание выравнивания не будет иметь никакого эффекта; оно имеет смысл, только когда свойству *Stretch* задано значение *None*. Далее представлен пример выравнивания изображения по правому краю и низу:

```
<Rectangle Stroke="Black" Width="200" Height="128" x:Name="r1">
  <Rectangle.Fill>
    <ImageBrush ImageSource="smily.jpg" Stretch="None"
                AlignmentX="Right" AlignmentY="Bottom" />
  </Rectangle.Fill>
</Rectangle>
```

Как это выглядит, можно увидеть на рис. 4-15.

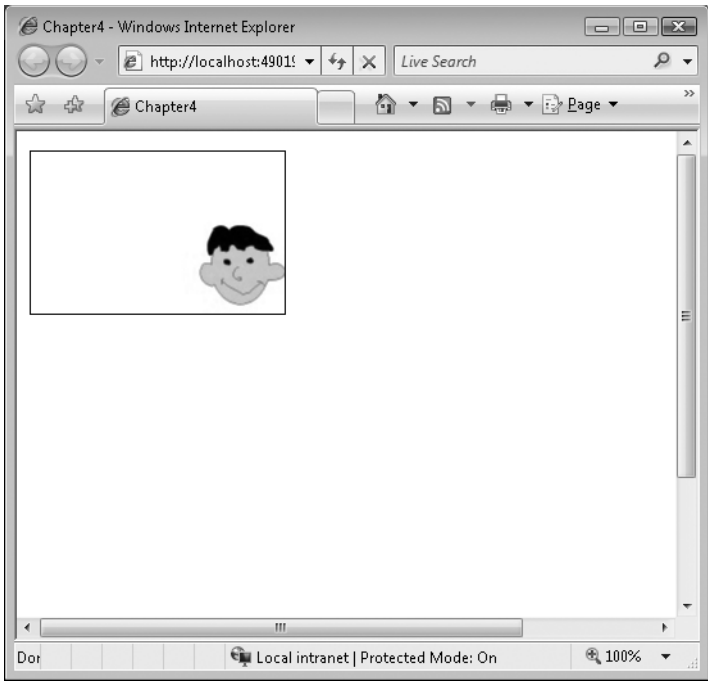

РИС. 4-15 Использование выравнивания изображения.

VideoBrush

VideoBrush (Видео-кисть) позволяет заполнять область видео-потоком. Более подробно эта кисть будет обсуждаться в Главе 10, «Мультимедиа в Silverlight: видео», в которой также более полно будет описан элемент управления *MediaElement*.

Визуальные свойства XAML

Кроме кистей и параметров расположения, XAML предоставляет множество других свойств для управления представлением объекта. Они позволяют задавать размеры, прозрачность объекта, поведение курсора и обводку.

Использование свойств XAML для задания размеров и местоположения

Размеры в XAML задаются с помощью свойств *Height* и *Width*, которые принимают значения типа *double*. Чтобы создать прямоугольник шириной 100 пикселов и высотой 200 пикселов, например, необходимо было бы написать следующий XAML-код:

```
<Rectangle Fill="Black" Width="100" Height="200" />
```

Кроме того, помните, что относительное местоположение объекта задается через точку с помощью свойств *Top* и *Left* в родительском элементе.

Рассмотрим следующий XAML:

```
<Canvas>
  <Canvas Canvas.Top="40" Canvas.Left="40">
    <Rectangle Canvas.Top="40" Fill="Black" Width="100" Height="200" />
  </Canvas>
</Canvas>
```

Предположим, что самый внешний элемент *Canvas* является корневым холстом страницы. В результате, *Rectangle* будет отрисован на расстоянии 80 пикселов от верха страницы, потому что его родительский холст опущен на 40 пикселов вниз, и *Rectangle* расположен на 40 пикселов вниз от верхнего края своего родителя, что в сумме дает 80 пикселов.

Использование прозрачности

Прозрачность объекта может быть задана двумя способами. Первый – с помощью альфа-канала кисти, которая используется для заливки объекта. Следующий XAML-код обеспечит создание черного прямоугольника поверх рисунка:

```
<Image Source="smily.jpg" />
<Rectangle Fill="#FF000000" Width="100" Height="200" />
```

В качестве значения свойства *Fill* задается черный цвет (потому что красному, зеленому и синему каналам задан 0), и альфа-канал задан непрозрачным (значение #FF). Меняя значение канала альфа, можно сделать прямоугольник полупрозрачным,:

```
<Image Source="smily.jpg" />
<Rectangle Fill="#77000000" Width="100" Height="200" />
```

Теперь прямоугольник будет выглядеть серым, и сквозь него будет видно изображение.

Второй метод – использовать свойство *Opacity* (Непрозрачность), которое принимает значения в диапазоне от 0 (полностью прозрачный) до 1 (полностью непрозрачный). Это свойство применяется в сочетании с определением альфа-канала кисти. Если для заливки фигуры использовать кисть цвета #77000000, например, и затем задать свойству *Opacity* значение 1, прямоугольник все равно будет непрозрачным. Если задать свойству *Opacity* значение 0, прямоугольник будет совершенно прозрачным.

Свойство *Opacity* удобно использовать для анимации прозрачности объекта. Оно позволяет без труда реализовать эффект постепенного исчезновения и появления объекта через *DoubleAnimation*. Более подробно анимация рассматривается в Главе 5, «Трансформация и анимация XAML».

Поведение курсора

Большинство элементов XAML позволяют использовать свойство *Cursor* (Курсор) для задания вида курсора при его прохождении по элементу. Это свойства принимает значения перечисления *MouseCursor* (Курсор мыши). Вот ряд наиболее часто используемых значений:

- ***Arrow (Стрелка)*** Отображается обычный стандартный курсор-стрелка
- ***Default (Стандартный)*** Нет предпочтительного курсора; используется курсор родительского элемента, если таковой задан
- ***Hand (Рука)*** Отображается курсор-рука, обычно используемый для ссылок
- ***IBeam (I-образный)*** Определяет I-образный курсор; обычно используется для выбора текста
- ***None (Нет)*** Курсор отсутствует
- ***Wait (Ожидание)*** Задает значок, который обозначает состояние ожидания

Управление свойством *Stroke*

Свойство *Stroke* определяет, как контур фигуры отрисовывается на экране. Обводка фигуры отличается от ее *заливки*.

Stroke задается с помощью кисти. Далее представлен пример XAML-кода, в котором прорисовка прямоугольника формируется черным контуром, заданным с помощью простой обводки:

```
<Rectangle Stroke="Black" Canvas.Left="40" Canvas.Top="40" Width="100" Height="200">
```

Свойство *Stroke*, в данном случае, использует *SolidColorBrush* черного цвета (*Black*). Синтаксически это эквивалентно следующему XAML:

```
<Rectangle Canvas.Left="40" Canvas.Top="40" Width="100" Height="200">
  <Rectangle.Stroke>
    <SolidColorBrush Color="Black" />
  </Rectangle.Stroke>
</Rectangle>
```

Используя такой синтаксис (определение кисти в качестве присоединенного свойства *Stroke*), можно задавать разные типы кистей для отрисовки обводки фигуры. Далее приведен пример применения *LinearGradientBrush* для обводки прямоугольника:

```
<Rectangle StrokeThickness="10" Canvas.Left="40"
        Canvas.Top="40" Width="100" Height="200">
  <Rectangle.Stroke>
    <LinearGradientBrush >
      <GradientStop Color="#FF000000" Offset="0"/>
      <GradientStop Color="#FFFFFFFF" Offset="0.5"/>
      <GradientStop Color="#FF000000" Offset="1"/>
    </LinearGradientBrush>
  </Rectangle.Stroke>
</Rectangle>
```

Как это выглядит на экране можно увидеть на рис. 4-16.

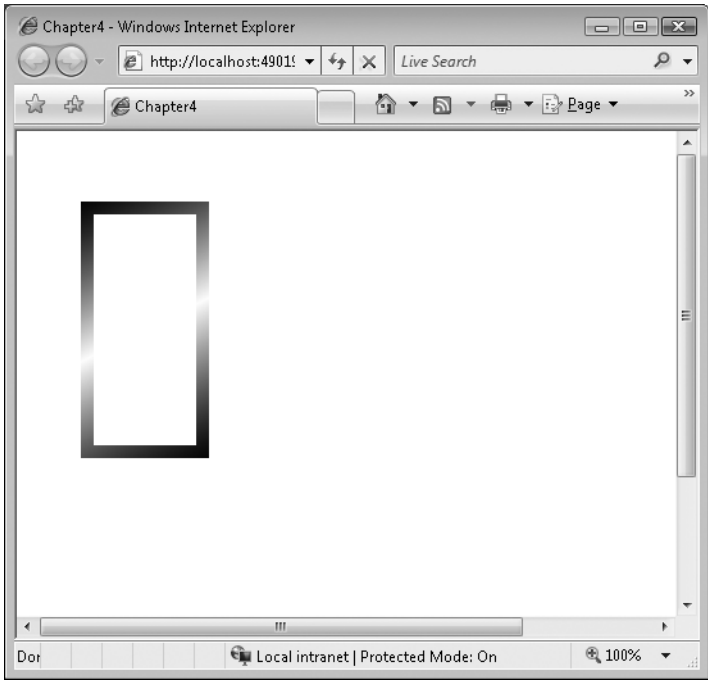

РИС. 4-16 Использование *LinearGradientBrush* для определения обводки фигуры.

Задание ширины обводки

Возможно, вы заметили, что в этом примере была задана ширина обводки 10. Это сделано для демонстрации градиентного изменения цвета, поскольку при использовании значения толщины по умолчанию, 1, его просто не было бы видно.

Ширина обводки задается свойством *StrokeThickness* (Толщина обводки). Оно определяет ширину обводки в пикселах:

```
<Rectangle StrokeThickness="10" Stroke="Black" Canvas.Left="40" Canvas.Top="40" Width="100" Height="200" />
```

Задание обводки пунктиром

Свойство *StrokeDashArray* (Массив длин элементов обводки пунктиром) используется для задания шаблона обводки пунктиром. В сочетании со свойствами *StrokeDashCap* (Наконечники элементов обводки пунктиром) и *StrokeDashOffset* (Смещение обводки пунктиром) оно обеспечивает детальное описание обводки пунктиром.

Чтобы обводка прямоугольника была выполнена пунктиром по определенному повторяющемуся шаблону, задается массив значений типа *double*, которые представляют длины чередующихся штрихов и пробелов. Чтобы определить шаблон пунктира с длиной штриха 4 единицы, за которым следует пробел длиной 1 единица, штрих длиной 2 единицы и пробел длиной 1 единица, свойству *StrokeDashArray* было бы задано значение (4,1,2,1). Вот пример:

```
<Rectangle StrokeThickness="10" Stroke="Black" Canvas.Left="40" Canvas.Top="40" Width="100" Height="200"
StrokeDashArray="4,1,2,1"/>
```

На рис. 4-17 показано, как это выглядело бы на экране.

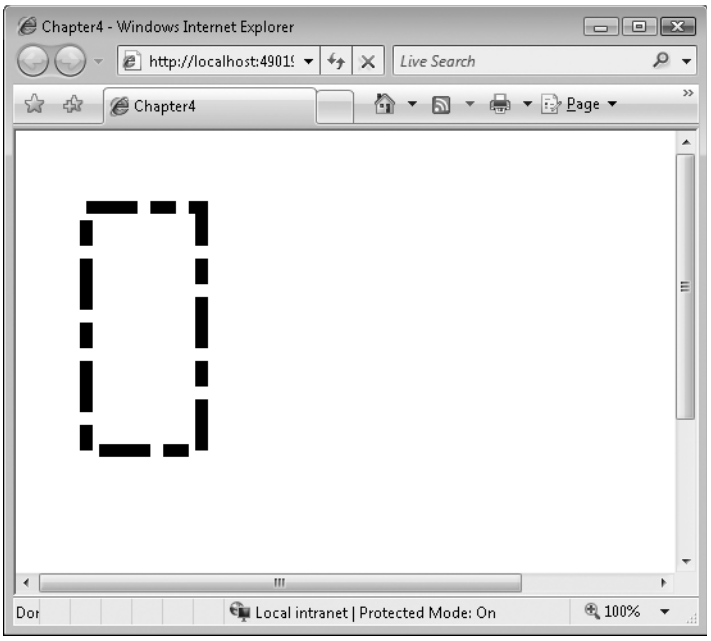

РИС. 4-17 Задание шаблона обводки пунктиром с помощью свойства *StrokeDashArray*.

Как видим, штрихи обводки имеют прямоугольную форму с нескошенными кромками. Изменить это представление можно с помощью свойства *StrokeDashCap*. В качестве его значений используется значения перечисления *PenLineCap* (Наконечники линий):

- ***Flat (Плоский)*** Это значение по умолчанию; оно определяет, что наконечник не выходит за край линии – аналогично отсутствию наконечника.
- ***Round (Круглый)*** Задает полукруг, диаметр которого равен толщине линии.
- ***Square (Квадратный)*** Определяет квадратный наконечник.
- ***Triangle (Треугольный)*** Определяет равнобедренный треугольный наконечник, базовая длина которого равна толщине обводки.

Далее представлен пример использования *StrokeDashCap* для задания скругленных штрихов обводки пунктиром:

```
<Rectangle StrokeThickness="10" Stroke="Black" Canvas.Left="40" Canvas.Top="40" Width="100" Height="200"
StrokeDashArray="4,1,2,1" StrokeDashCap="Round"/>
```

На рис. 4-18 показано, как это будет выглядеть на экране.

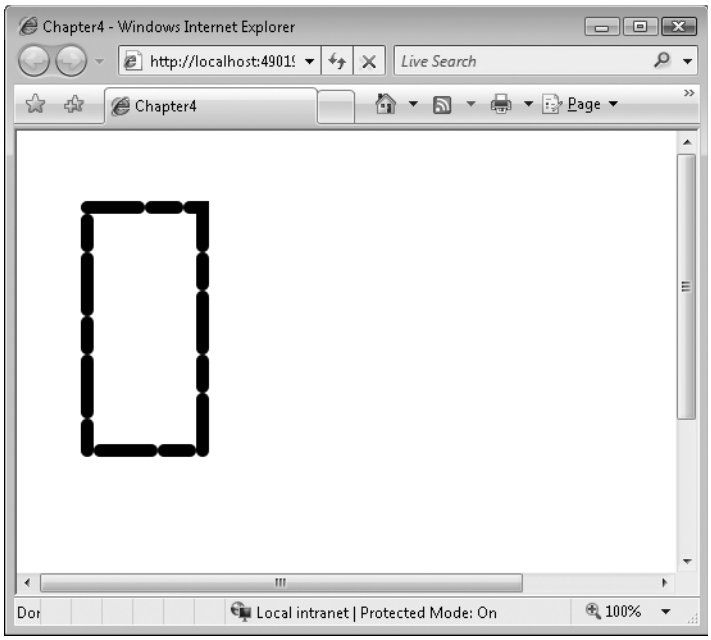

РИС. 4-18 Использование свойства *StrokeDashCap*.

Определение типа соединения линий

Если взглянуть на предыдущие примеры, можно заметить, что штрихи пунктира пересекаются в углах и отрисовываются спрямленными. Это считается соединением линий. С помощью свойства *StrokeLineJoin* (Соединение линий обводки) можно определять поведение пера при соединении линий. Это свойство принимает перечисление *PenLineJoin*, которое может содержать одно из следующих трех значений:

- ***Bevel (Скос)*** Срезает края соединения
- ***Miter (Прямой угол)*** Сохраняет острые края
- ***Round (Скругление)*** Скругляет края соединения

Данный XAML-код создает прямоугольник, для которого задан тип соединения линий *Bevel*:

```
<Rectangle StrokeThickness="10" Stroke="Black" Canvas.Left="40" Canvas.Top="40" Width="100" Height="200"
StrokeLineJoin="Bevel" />
```

Как это отображается, можно увидеть на рис. 4-19.

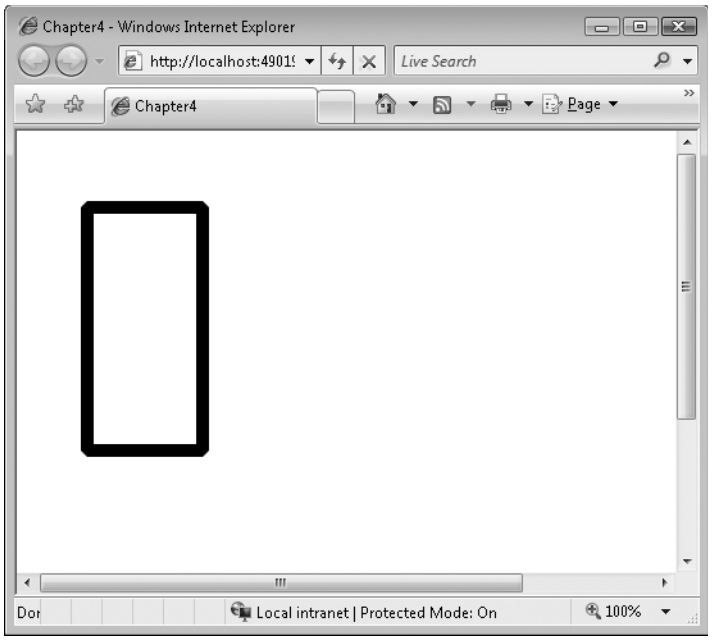

РИС. 4-19 Использование *StrokeLineJoin* для задания срезанных углов.

Фигуры в XAML

XAML поддерживает ряд базовых фигур, которые могут использоваться для создания более сложных объектов. К ним относятся:

- ***Ellipse (Эллипс)*** Отрисовывает эллипс; окружность – это эллипс с равными радиусами по осям *X* и *Y*
- ***Rectangle (Прямоугольник)*** Отрисовывает прямоугольник; квадрат – это прямоугольник с равными размерами по осям *X* и *Y*
- ***Line (Линия)*** Отрисовывает прямую линию
- ***Path (Контур)*** Отрисовывает последовательность соединенных прямых и кривых линий соответственно языку описания контуров
- ***Polygon (Многоугольник)*** Отрисовывает замкнутую фигуру, образованную рядом соединенных линий
- ***Polyline (Ломаная)*** Отрисовывает ряд последовательно соединенных прямых

В следующих разделах мы рассмотрим каждую из этих фигур.

Использование объекта *Ellipse*

Фигура *Ellipse* используется для отрисовки эллипса или круга. Высота *Ellipse* определяется вертикальным диаметром (т.е. двойным радиусом по вертикали), его ширина определяется горизонтальным диаметром. Если эти значения равны, XAML сформирует круг. Контур *Ellipse* отрисовывается с помощью свойства *Stroke* и закрашивается с помощью *Brush*.

Использование объекта *Rectangle*

Rectangle используется для отрисовки прямоугольника или квадрата. Размеры данной фигуры задаются свойствами *Width* и *Height*. Если значения этих свойств равны, получаем квадрат. Как и для *Ellipse*, контур *Rectangle* отрисовывается с помощью свойства *Stroke*, для заливки используется *Brush*.

Используя свойства *RadiusX* и *RadiusY*, можно скруглить углы *Rectangle*. Задавая этим свойствам значения типа *double*, вы определяете необходимый радиус скругления. По умолчанию они равны 0.0, что указывает на отсутствие скругления. Поскольку свойства *RadiusX* и *RadiusY* задаются независимо друг от друга, применяя разные значения для них, можно получить эллиптические скругления.

Далее представлен пример прямоугольника со скругленными углами. Свойству *RadiusY* задано значение 40, и свойству *RadiusX* – значение 20. Это указывает на то, что углы будут более пологими в вертикальном направлении:

```
<Rectangle Fill="Black" Canvas.Left="40" Canvas.Top="40"
    Width="100" Height="200" RadiusX="20" RadiusY="40" />
```

Результаты задания таких значений в действии можно увидеть на рис. 4-20.

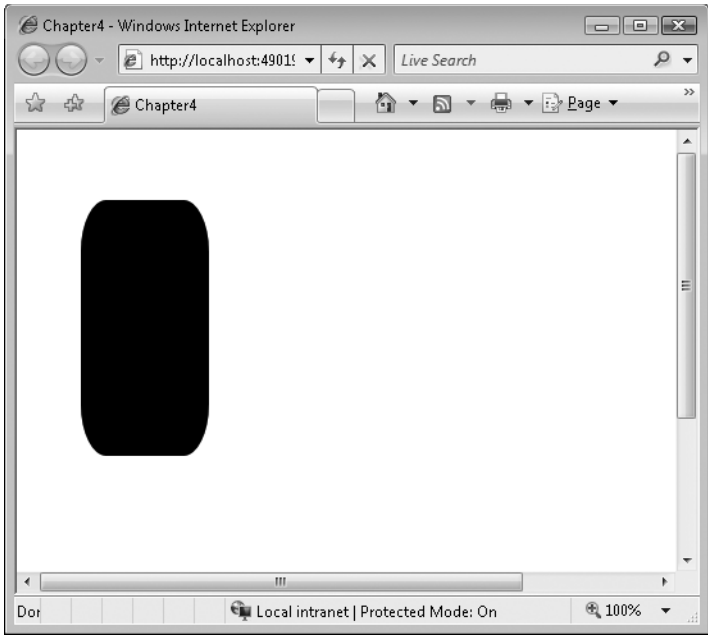

РИС. 4-20 Скругление углов прямоугольника.

Использование объекта *Line*

С помощью объекта *Line* можно создать простую линию в XAML. Для этого задаются координаты (X1,Y1) и (X2,Y2), и линия отрисовывается между этими точками. Данные координаты задаются относительно верхнего левого угла области, который определяется свойствами *Canvas.Top* и *Canvas.Left*, как обычно. Обратите внимание, что чтобы линия была отрисована, обязательно должна быть определена обводка, хотя бы ее цвет.

Рассмотрим следующий XAML:

```
<Line X1="40" Y1="40" X2="100" Y2="100" Stroke="Black" />
```

Это обеспечит отрисовку линии, соединяющей точку (40,40) с точкой (100,100). Однако если добавить свойства *Canvas.Top* и/или *Canvas.Left*, линия будет отрисовываться со смещением. Если предположить, что родительский холст не позиционирован, следующий XAML обеспечит создание линии, соединяющей точки (40,140) и (100,200). Если позиционирование присутствует, Silverlight отрисует линию, относительно позиции родительского элемента.

```
<Line Canvas.Top="100" X1="40" Y1="40" X2="100" Y2="100" Stroke="Black" />
```

Использование контуров и геометрических элементов

Объект *Path* отрисовывает последовательность соединенных прямых и кривых линий. Эти прямые и кривые определяются типом геометрического элемента. В данном разделе рассматриваются различные типы геометрических элементов.

EllipseGeometry (Геометрический элемент эллипс) определяет контур как простой эллипс. Например:

```
<Path Stroke="Black">
  <Path.Data>
    <EllipseGeometry RadiusX="100" RadiusY="100" />
  </Path.Data>
</Path>
```

Размеры эллипса, образующего контур, задаются в *EllipseGeometry* с помощью свойств *RadiusX* и *RadiusY*. Также с помощью атрибута *Center* (Центр) можно задать центр эллипса. Например:

```
<Path Stroke="Black">
   <Path.Data>
      <EllipseGeometry RadiusX="100" RadiusY="100" Center="50,50" />
   </Path.Data>
</Path>
```

LineGeometry (Геометрический элемент прямая) определяет контур как одну линию, начинающуюся в *StartPoint*, и заканчивающуюся в *EndPoint*. Эти точки задаются с помощью простых координат x и y, таким образом, верхний левый угол задается как (0,0). Вот пример *LineGeometry*:

```
<Path Stroke="Black">
  <Path.Data>
    <LineGeometry StartPoint="10,10" EndPoint="100,100" />
  </Path.Data>
</Path>
```

RectangleGeometry определяет контур как отдельный прямоугольник. Размеры прямоугольника задаются свойством *Rect*. Это строка из четырех значений, соответствующих отступу сверху, отступу слева, высоте и ширине прямоугольника. Итак, чтобы отрисовать прямоугольник 100 на 200 пикселов, верхний левый угол которого располагается в точке (0,0), использовался бы следующий *Path*:

```
<Path Stroke="Black">
  <Path.Data>
    <RectangleGeometry Rect="0,0,100,200" />
  </Path.Data>
</Path>
```

Объект *PathGeometry* (Геометрический элемент контур) используется для задания сложного контура, состоящего из разных сегментов, включая дуги, кривые Безье, прямые линии, кривые Безье высших степеней, поликвадратические кривые Безье и квадратические кривые Безье. Сегменты образуют *PathFigure* (Форма контура), и один или более объектов *PathFigure* образуют *PathGeometry*. *PathGeometry* также задает начальную точку контура. Если имеется множество сегментов, начальной точкой каждого последующего сегмента будет конечная точка предыдущего сегмента.

Объект *ArcSegment*

Объект *ArcSegment* (Сегмент дуга) отрисовывает простую эллиптическую дугу между двумя точками. Дуга описывается рядом разнообразных свойств:

- ***Point (Точка)*** Задает начальную точку дуги
- ***Size (Размер)*** Задает радиус дуги по осям x и y
- ***RotationAngle (Угол поворота)*** Задает угол поворота, т.е. то, насколько сильно дуга закручена вокруг оси x

- ***IsLargeArc (Является ли дуга крупной)*** Определяет, является ли дуга крупной, где крупной считается дуга более 180°
- ***SweepDirection (Направление развертывания)*** Задает направление отрисовки дуги (*Clockwise* (По часовой стрелке) или *CounterClockwise* (Против часовой стрелки))

Вот пример *Path* с одним дуговым сегментом, в котором демонстрируются данные свойства:

```
<Path Stroke="Black">
  <Path.Data>
    <PathGeometry>
      <PathFigure>
        <ArcSegment Point="100,100" Size="200,200"
          RotationAngle="10" IsLargeArc="False"
          SweepDirection="ClockWise" />
      </PathFigure>
    </PathGeometry>
  </Path.Data>
</Path>
```

Объект *LineSegment*

Добавить прямую линию в *PathSegment* (Сегмент контура) можно с помощью объекта *LineSegment* (Сегмент линия). Он просто отрисовывает прямую линию от текущей или начальной точки до точки, заданной свойством *Point*. Итак, чтобы отрисовать прямую из точки (100,100) до точки (200,200) и затем еще одну до точки (200,0), создается *PathFigure*, содержащий несколько линейных сегментов:

```
<Path Stroke="Black">
  <Path.Data>
    <PathGeometry>
      <PathFigure StartPoint="100,100">
        <LineSegment Point="200,200" />
        <LineSegment Point="200,0" />
      </PathFigure>
    </PathGeometry>
  </Path.Data>
</Path>
```

Объект *PolyLineSegment*

Как следует из его имени, с помощью объекта *PolyLineSegment* (Сегмент ломаная) можно отрисовать ряд прямых, просто задав точки. Первая линия отрисовывается между начальной точкой и первой заданной точкой, вторая – между этой точкой и второй заданной точкой, и т.д.

Далее приведен XAML-код, демонстрирующий *PolyLineSegment*:

```
<Path Stroke="Black">
  <Path.Data>
    <PathGeometry>
        <PathFigure StartPoint="100,100">
          <PolyLineSegment
           Points="50,50,150,150,250,250,
                   100,200,200,100,300,300" />
        </PathFigure>
    </PathGeometry>
  </Path.Data>
</Path>
```

Объект *BezierSegment*

Этот объект позволяет описывать кривую Безье, которая является кривой между двумя точками, определяемой одной или двумя опорными точками. Объект *BezierSegment* (Сегмент кривая Безье) принимает в качестве параметров три точки, называемые *Point1*, *Point2* и *Point3*. В зависимости от того, сколько точек используется, вы получаете разное поведение. Итак, например, если заданы *Point1* и *Point2*, кривая будет сформирована от начальной точки в точку *Point2* с использованием *Point1* в

качестве опорной точки. Если заданы *Point1*, *Point2* и *Point3*, кривая формируется от начальной точки в точку *Point3*, *Point1* и *Point2* используются как опорные.

Вот пример *PathFigure*, содержащего *BezierSegment*:

```
<Path Stroke="Black">
  <Path.Data>
    <PathGeometry>
      <PathFigure StartPoint="100,100">
        <BezierSegment Point1="140,120" Point2="100,140" />
      </PathFigure>
    </PathGeometry>
  </Path.Data>
</Path>
```

Объект *PolyBezierSegment*

Этот объект позволяет задавать группу точек, которые Silverlight интерпретирует в набор опорных точек для группы кривых Безье. Рассмотрим следующий XAML:

```
<Path Stroke="Black">
 <Path.Data>
  <PathGeometry>
   <PathFigure StartPoint="100,100">
    <PolyBezierSegment>
     <PolyBezierSegment.Points>
      <Point X="50" Y="50" />
      <Point X="150" Y="150" />
      <Point X="250" Y="250" />
      <Point X="100" Y="200" />
      <Point X="200" Y="100" />
      <Point X="300" Y="300" />
     </PolyBezierSegment.Points>
    </PolyBezierSegment>
   </PathFigure>
  </PathGeometry>
 </Path.Data>
</Path>
```

Как альтернативный вариант, можно задать ряд точек с помощью коллекции точек, хранящейся в виде строки значений, разделенных запятыми:

```
<Path Stroke="Black">
  <Path.Data>
    <PathGeometry>
      <PathFigure StartPoint="100,100">
        <PolyBezierSegment  Points="50,50,150,150,250,250,
                          100,200,200,100,300,300" />
      </PathFigure>
    </PathGeometry>
  </Path.Data>
</Path>
```

Визуальное представление этого XAML-фрагмента показано на рис. 4-21. В данном случае, начальная точка определена координатами (100,100), это верхняя левая точка всей кривой. Первая кривая Безье заканчивается в точке (250,250), используя точки (50,50) и (150,150) в качестве опорных. Поскольку эти точки эффективно отменяют действие друг друга (они равноудалены от линии, соединяющей точки [100,100] и [250,250]), первая кривая Безье является прямой линией, оканчивающейся в точке (250,250). Вторая кривая Безье начинается в этой точке и заканчивается в точке (300,300) с опорными точками (100,200) и (200,100), благодаря чему она принимает вид петли.

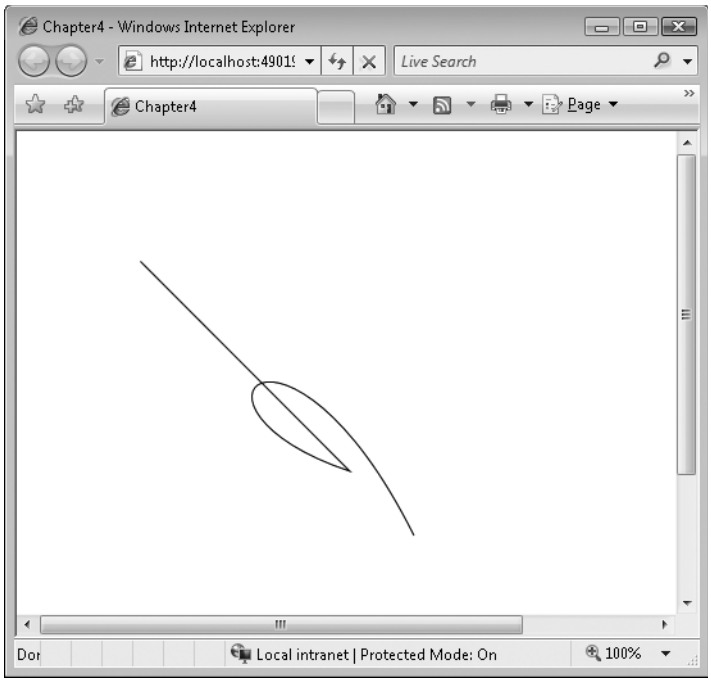

РИС. 4-21 Использование *PolyBezierSegment*.

Объект *QuadraticBezierSegment*

Квадратическая кривая Безье – это просто кривая Безье, построенная как обычная квадратическая кривая с использованием единственной опорной точки. *QuadraticBezierSegment* (Сегмент квадратическая кривая Безье) принимает два объекта *Point*. Если используется только один объект, он становится конечной точкой кривой. В этом случае контрольной точки нет, и кривая принимает форму прямой линии, соединяющей начальную и заданную конечную точки. Если задается две точки, вторая точка принимается за конечную точку кривой, а первая точка становится квадратической опорной точкой. Вот пример:

```
<Path Stroke="Black">
  <Path.Data>
    <PathGeometry>
      <PathFigure StartPoint="100,100">
        <QuadraticBezierSegment Point1="200,0" Point2="300,100"  />
      </PathFigure>
    </PathGeometry>
  </Path.Data>
</Path>
```

Кривая, формируемая данным XAML, представлена на рис. 4-22.

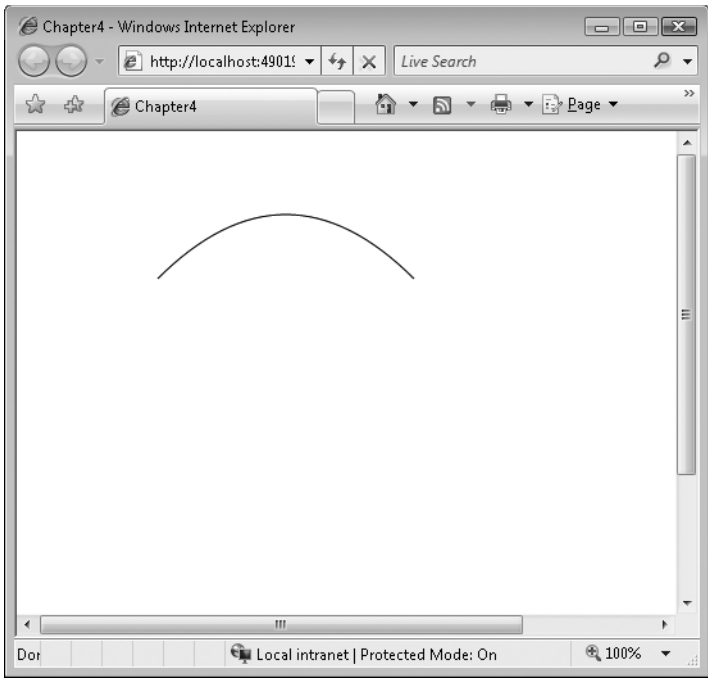

РИС. 4-22 Простая квадратическая кривая Безье.

Объект *PolyQuadraticBezierSegment*

Как следует из его имени, *PolyQuadraticBezierSegment* (Сегмент поликвадратическая кривая Безье) является коллекцией соединенных квадратических кривых Безье, определенных и построенных соответственно списку опорных точек аналогично тому, как это описывалось ранее для объекта *PolyBezierSegment*. Далее представлен пример *PolyQuadraticBezierSegment* в действии:

```
<Path Stroke="Black">
  <Path.Data>
    <PathGeometry>
      <PathFigure StartPoint="100,100">
        <PolyQuadraticBezierSegment Points="50,50,150,150,250,250,
                                    100,200,200,100,300,300" />
      </PathFigure>
    </PathGeometry>
  </Path.Data>
</Path>
```

Визуальное представление данного XAML-фрагмента показано на рис. 4-23.

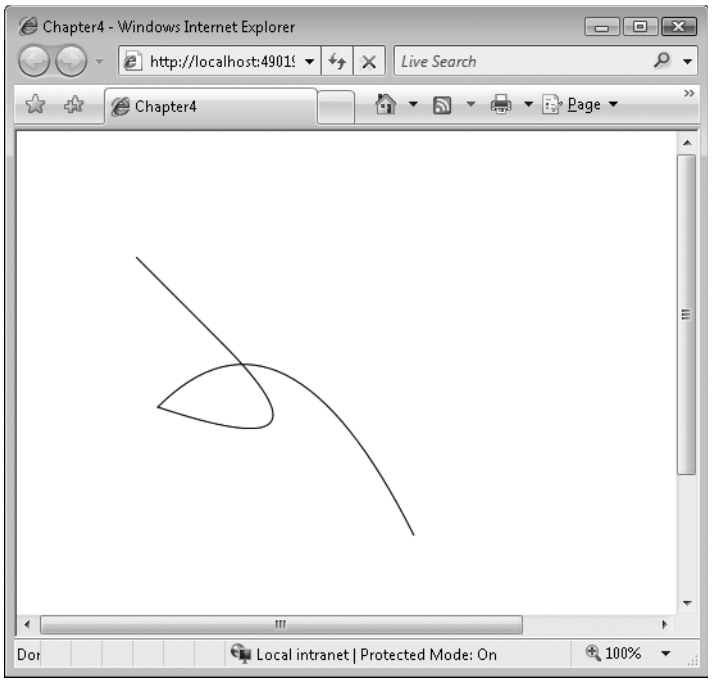

РИС. 4-23 Объект в действии *PolyQuadraticBezierSegment*.

Данный код обеспечивает построение ряда квадратических кривых Безье. Первая строится между начальной точкой, (100,100), и точкой (150,150) с использованием точки (50,50) в качестве опорной, что в результате дает прямую линию, оканчивающуюся в точке (150,150). Вторая кривая начинается в точке (150,150) и заканчивается в точке (100,200) с опорной точкой (250,250). Это сегмент, в котором кривая разворачивается влево. Третья кривая начинается в точке (100,200) и заканчивается в точке (300,300) с опорной точкой (200,100). На рис. 4-23 это длинная сглаженная кривая, идущая слева направо.

Составные сегменты контура

Каждый из рассмотренных типов сегментов может входить в состав сегмента *PathFigure*. Кроме того, сегменты *PathFigure* могут собираться в коллекции в рамках *PathGeometry* для создания сложного набора сегментов.

Далее приведен пример *PathGeometry*, в состав которого входят два объекта *PathFigure*. Первый объект содержит *LineSegment*, *PolyQuadraticBezierSegment* и еще один *LineSegment*. Второй объект включает только один *LineSegment*.

```
<Path Stroke="Black">
  <Path.Data>
    <PathGeometry>
      <PathFigure StartPoint="100,100">
        <LineSegment Point="200,200" />
        <PolyQuadraticBezierSegment Points="50,50,150,150,250,250,
                                    100,200,200,100,300,300" />
        <LineSegment Point="0,0" />
      </PathFigure>
      <PathFigure>
        <LineSegment Point="10,400" />
      </PathFigure>
    </PathGeometry>
  </Path.Data>
</Path>
```

Использование объекта *GeometryGroup*

В предыдущих разделах были представлены всевозможные геометрические элементы, начиная от самых простых, таких как *EllipseGeometry*, *LineGeometry* и *RectangleGeometry*, до сложных, образованных

множеством объектов *PathSegment* в рамках *PathGeometry*. Объект *GeometryGroup* (Группа геометрических элементов) позволяет комбинировать все эти объекты.

Вы просто описываете геометрические элементы, которые хотите включить как коллекцию в этот объект. Далее представлен объект *GeometryGroup*, содержащий *EllipseGeometry*, *RectangleGeometry*, и затем тот же сложный *PathGeometry*, который использовался в предыдущем разделе:

```
<Path Stroke="Black">
  <Path.Data>
    <GeometryGroup>
      <EllipseGeometry RadiusX="100" RadiusY="100" Center="50,50" />
      <RectangleGeometry Rect="200,200,100,100" />
      <PathGeometry>
        <PathFigure StartPoint="100,100">
          <LineSegment Point="200,200" />
          <PolyQuadraticBezierSegment Points="50,50,150,150,250,250,
                                      100,200,200,100,300,300" />
          <LineSegment Point="0,0" />
        </PathFigure>
        <PathFigure>
          <LineSegment Point="10,400" />
        </PathFigure>
      </PathGeometry>
    </GeometryGroup>
  </Path.Data>
</Path>
```

РИС. 4-24 представляет, как это будет выглядеть.

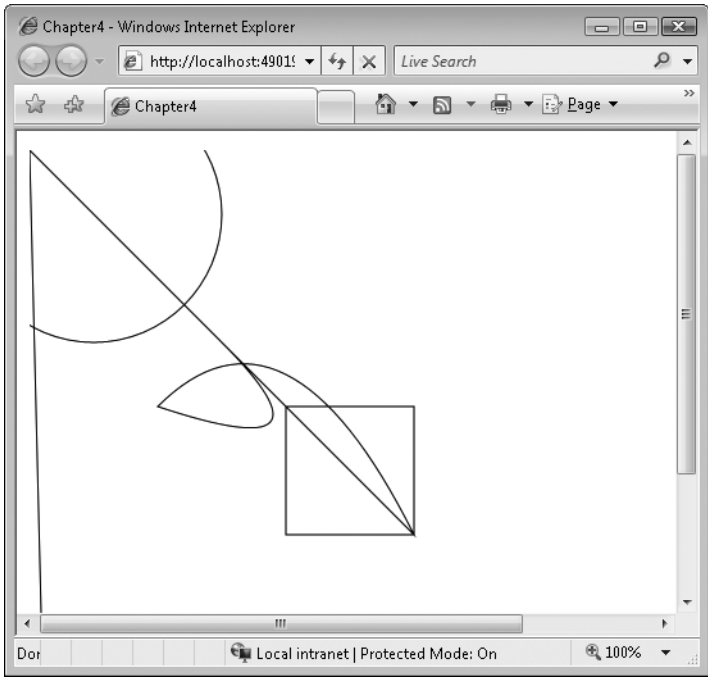

РИС. 4-24 Использование объекта *GeometryGroup* для группировки множества геометрических элементов.

Язык описания контуров

Объект *Path* имеет свойство *Data* (Данные), в котором может использоваться язык *описания контуров* для определения сложных контуров. В этом языке используется следующий синтаксис: буква команды, за ней через пробел следует список разделенных запятыми чисел, за которым через пробел идет следующая буква команды. Вот пример:

```
<Path Stroke="Black" Data="M 100,100 L 200,200" />
```

- В этом экземпляре контур отрисовывается с помощью команды *M*, обозначающей *Move* (Перейти), для перехода в точку (100,100) и команды *L*, обозначающей *Line*, для построения прямой линии до

точки (200,200). Эти команды могут быть заданы либо прописными буквами (например, *M*), что свидетельствует о применении абсолютных координат, или строчными буквами (например, *m*), что определяет относительные координаты. Команда M помещает перо в заданную точку без отрисовки линии между точкой, в которой он находится в настоящий момент, и заданной точкой; т.е. перо перемещается в начальную точку. Затем будет проведена линия из точки (100,100) в точку (200,200), при условии отсутствия позиционирования родительского холста. Если родительский холст позиционирован, Silverlight отрисует линию относительно его положения.

- Команда *L* (означающая Line) обеспечивает построение прямой линии от текущей точки в заданную.
- Команда *H* принимает в качестве параметра только одно число и отрисовывает горизонтальную прямую между текущей точкой и заданным значением по оси x.
- Команда *V* принимает в качестве параметра только одно число и отрисовывает вертикальную прямую между текущей точкой и заданным значением по оси y.
- Команда *C* принимает в качестве параметров три точки. Она обеспечивает построение кубической кривой Безье между текущей точкой и третьей из заданных точек. Две первые точки используются как опорные.
- Команда *Q* принимает в качестве параметров две точки и отрисовывает квадратическую кривую Безье между текущей точкой и второй из заданных точек. Первая заданная точка используется как опорная.
- Команда *S* принимает в качестве параметров две точки и отрисовывает сглаженную кубическую кривую Безье между текущей точкой и второй из заданных точек. Для формирования сглаженной кривой используется две опорные точки: текущая точка и первый из двух параметров.
- Команда *T* действует аналогично команде *S*, за исключением того что она обеспечивает построение сглаженной квадратической кривой Безье.
- Команда *A* принимает пять параметров: размер, угол поворота, *isLargeArc*, *sweepDirection* и конечную точку. Эти параметры используются для построения эллиптической дуги.
- Команда *Z* завершает текущий контур, проводя прямую линию между текущей точкой и начальной точкой контура и образуя замкнутую фигуру.

Вырезание и геометрические элементы в XAML

Вырезание элементов в XAML осуществляется по правилу, определяемому типом геометрической формы. Как было показано в предыдущем разделе, эти типы могут быть простыми (*EllipseGeometry, LineGeometry* или *RectangleGeometry*), сложными (использующими геометрическую форму, определенную объектом *PathGeometry*) или группой из этих объектов в составе *GeometryCollection*.

Чтобы определить метод вырезания объекта, его присоединенному свойству *Clip* просто задается тип геометрической формы. Следующий пример представляет изображение, вырезанное *EllipseGeometry*:

```
<Image Source="smily.jpg" Width="300" Height="300">
<Image.Clip>
  <EllipseGeometry Center="150,150" RadiusX="100" RadiusY="100" />
</Image.Clip>

</Image>
```

Увидеть, как это выглядит, можно на рис. 4-25.

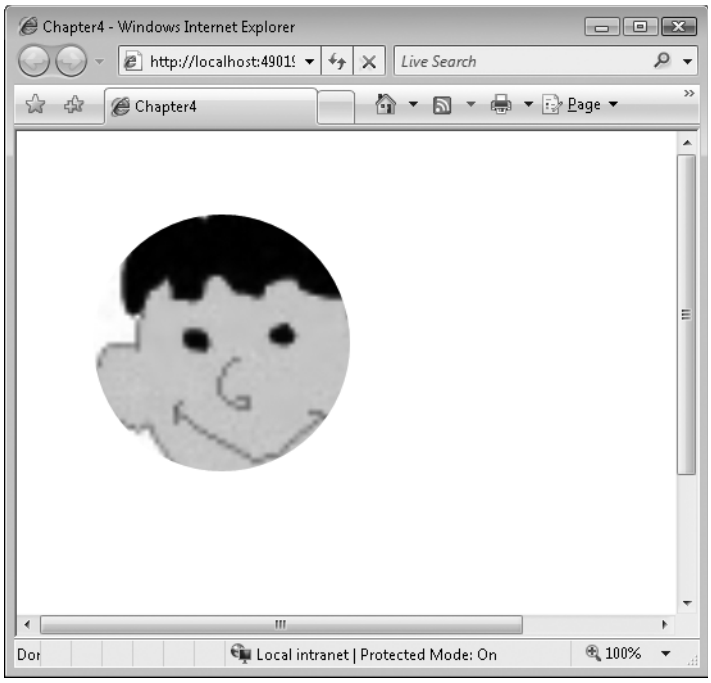

РИС. 4-25 Вырезание объекта геометрической формой.

Заключение

С помощью XAML можно описывать пользовательские интерфейсы. Данная глава рассматривает некоторые основные элементы XAML и их работу. Но это лишь самая верхушка айсберга. В последующих главах будут продемонстрированы более сложные варианты использования XAML, включая управление компоновкой элементов управления, а также конфигурацию самих элементов управления.

В данной главе подробно рассмотрена настройка визуальных элементов с помощью XAML. Представлены компоновка, позиционирование, заливка, обводки, прозрачность, контуры, геометрические элементы и вырезание – возможности, которые позволяют управлять тем, что вы видите в своем пользовательском интерфейсе.

Следующая глава будет посвящена использованию XAML для оживления пользовательского интерфейса через введение трансформаций и анимации!

Глава 5

XAML: трансформация и анимация

В Главе 4, «Основы Silverlight XAML», вы научились с помощью Extensible Application Markup Language (XAML) формировать визуальное представление графических элементов на экране, будь то векторная графика, растровые изображения (с помощью объекта *Image*) или видео. В данной главе будет рассмотрено, как улучшить представление этих графических элементов с помощью различных типов трансформаций (для изменения их вида) и анимации (для изменения атрибутов объекта во времени). Кроме того, будет представлена анимация по ключевым кадрам и описано их использование при создании более сложного поведения. В завершение мы опять ненадолго вернемся к Microsoft Expression Blend и покажем, как его можно использовать для создания анимации.

Трансформации

В графике трансформация определяет, как точки из одного координатного пространства отображаются в другом координатном пространстве. Обычно это описывается с помощью *матрицы трансформации* (*transformation matrix*), специальной математической структуры, которая обеспечивает простое математическое преобразование из одной системы в другую. Microsoft Silverlight XAML позволяет абстрагировать это, и в данной книге мы не будем вдаваться в математические детали. Silverlight XAML поддерживает четыре вида трансформации элементов: вращение, масштабирование, наклон и трансляцию (перемещение), а также особый тип трансформации, который позволяет реализовывать собственную матрицу для сочетания трансформаций.

Трансформации реализуются через свойства трансформации. Существует несколько разных типов свойств трансформации, применяемых к разным типам объектов.

Так при использовании типа *Brush* трансформация определяется либо через свойство *Brush.Transform (Трансформация кисти)*, если необходимо изменить содержимое кисти (например, если вы хотите повернуть изображение, прежде чем использовать его в *ImageBrush*), либо с помощью свойства *Brush.RelativeTransform (Относительная трансформация кисти)*, которое позволяет трансформировать кисть, используя относительные значения (это пригодиться при закрашивании одной кистью разных областей разных размеров).

Простые трансформации для типа *Geometry* выполняются с помощью свойства *Geometry.Transform*. Этот тип не поддерживает относительные трансформации.

Наконец, при использовании элемента пользовательского интерфейса (UI) трансформация задается через свойство *RenderTransform*. Например, трансформация эллипса будет выполняться с помощью свойства *Ellipse.RenderTransform*.

В следующем разделе будут рассмотрены разные типы трансформаций и показано использование этих свойств для конкретных типов объектов.

Вращение с помощью *RotateTransform*

RotateTransform позволяет поворачивать элемент на заданный угол вокруг заданной центральной точки. Угол поворота задается свойством *Angle* (Угол) в градусах. Примем за начало отсчета (0 градусов) горизонтальный вектор, направленный вправо. Пусть вращение выполняется *по часовой*

стрелке, тогда вертикальный вектор, указывающий вниз, является результатом поворота на 90 градусов.

Координаты центра трансформации задаются свойствами *CenterX* и *CenterY*. По умолчанию центр трансформации имеет координаты 0.0, таким образом, ось вращения по умолчанию находится в верхнем левом углу контейнера.

Рассмотрим пример XAML-кода, в котором выполняется вращение *TextBlock* с помощью *RenderTransform*, содержащего *RotateTransform*, который определяет поворот на 45 градусов:

```
<TextBlock Width="320" Height="40"
        Text="This is the text to rotate" TextWrapping="Wrap">
  <TextBlock.RenderTransform>
    <RotateTransform Angle="45" />
  </TextBlock.RenderTransform>
</TextBlock>
```

Как это выглядит, представлено на рис. 5-1.

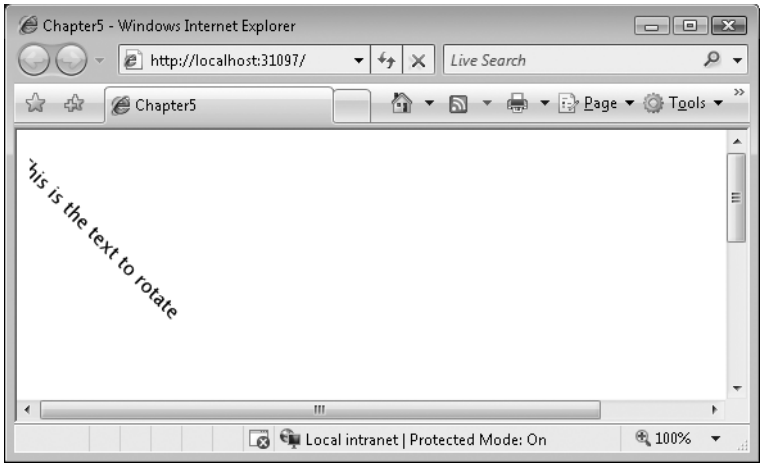

РИС. 5-1 Использование свойства *RotateTransform*.

Как видите, поворот этого текста был выполнен относительно центральной точки с координатами (0,0), т.е. верхнего левого угла.

В следующем XAML показано, как использовать *CenterX* и *CenterY* для выполнения вращения вокруг другой точки. В данном случае, поворот выполняется вокруг точки (100,200):

```
<TextBlock Width="320" Height="40"
        Text="This is the text to rotate" TextWrapping="Wrap" >
  <TextBlock.RenderTransform>
    <RotateTransform Angle="45" CenterX="100" CenterY="200" />
  </TextBlock.RenderTransform>
</TextBlock>
```

Результаты этой трансформации представлены на рис. 5-2.

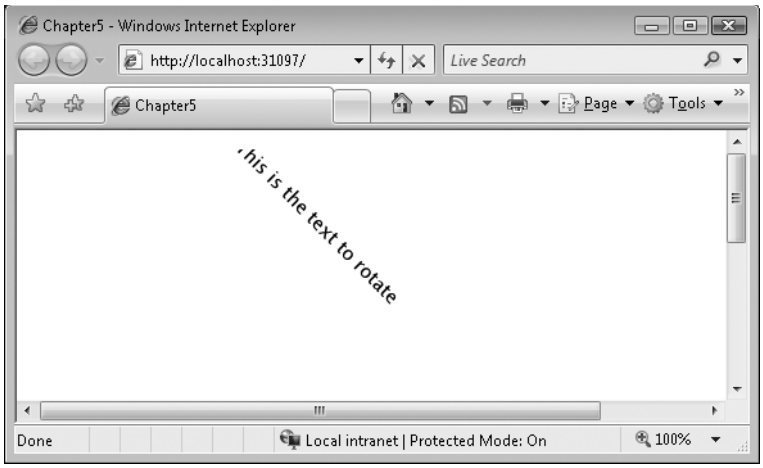

РИС. 5-2 Вращение вокруг другой центральной точки.

Масштабирование с помощью *ScaleTransform*

ScaleTransform используется для изменения размера объекта в направлении по горизонтальной оси, вертикальной оси или в обоих направлениях.

При выполнении масштабирования объекта необходимо задать, по крайней мере, одну из осей масштабирования и то, на сколько должен быть изменен масштаб объекта по этой оси. Для масштабирования объекта по горизонтальной оси, оси x, используется свойство *ScaleX*, и для масштабирования по вертикальной оси, оси y, используется свойство *ScaleY*. Эти свойства принимают значения типа *double*, которые представляют, во сколько раз должен быть изменен объект в направлении по заданной оси. Следовательно, значения, больше 1, будут увеличивать объект. Например, если свойству *ScaleX* задать значение 2, это обеспечит увеличение размера объекта по горизонтали в два раза. Значения, меньше 1, но больше 0, будут обусловливать уменьшение объекта. Например, использование значения 0,5 приведет к уменьшению размера объекта вдвое в заданном направлении.

Итак, рассмотрим XAML-код, который обеспечивает создание красного прямоугольника шириной 96 пикселов и высотой 88 пикселов:

```
<Rectangle Fill="#FFFF0404" Stroke="#FF000000" Width="96" Height="88"
Canvas.Left="112" Canvas.Top="72" />
```

На рис. 5-3 показано, как такой прямоугольник выглядит в Silverlight.

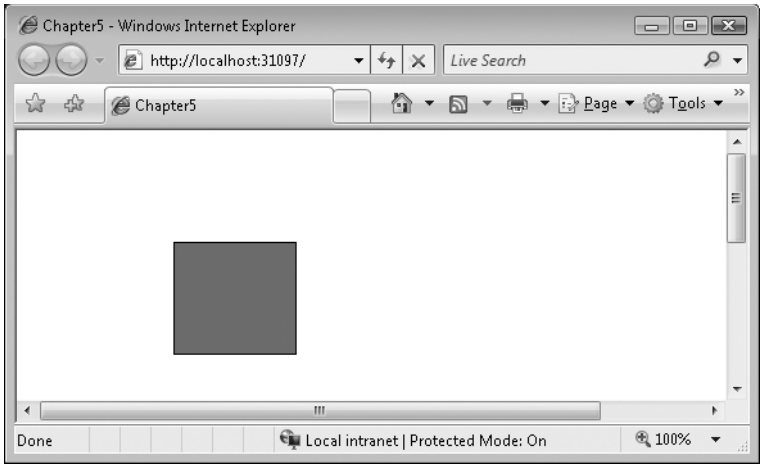

РИС. 5-3 Представление прямоугольника в Silverlight.

Чтобы применить к этому объекту *ScaleTransform*, используется *RenderTransform* и задается трансформация *ScaleTransform*. Вот соответствующий XAML:

```
<Rectangle Fill="#FFFF0404" Stroke="#FF000000"
      Width="96" Height="88" Canvas.Left="112" Canvas.Top="72">
  <Rectangle.RenderTransform>
    <ScaleTransform ScaleX="2" />
  </Rectangle.RenderTransform>
</Rectangle>
```

На рис. 5-4 показано, как это выглядит в Silverlight.

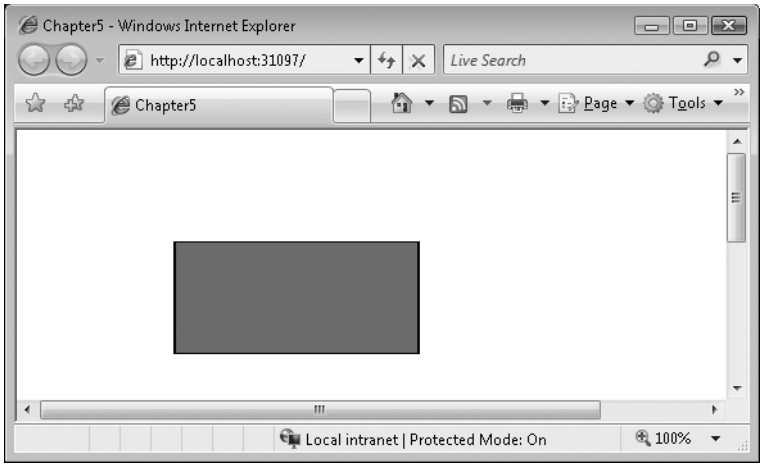

РИС. 5-4 Масштабирование с помощью *ScaleTransform*.

Можно заметить, что в результате применения *ScaleTransform* прямоугольник вытянулся вправо. Это произошло, потому что не был задан центр масштабирования. Для задания центра масштабирования используются свойства *CenterX*, для масштабировании по горизонтали, и *CenterY*, для масштабирования по вертикали. Эти свойства определяют координаты центра масштабирования. Эти координаты задаются относительно верхнего левого угла прямоугольника. Координата по умолчанию – 0, т.е. изменение размера будет происходить вправо и вниз.

Если свойству *CenterX* задать положительное значение (например, 50), масштабирование будет выполняться относительно точки X, смещенной на 50 пикселов вправо от левого края прямоугольника. В результате этого будет создаваться впечатление, что прямоугольник сместился на некоторое число пикселов (в зависимости от коэффициента масштабирования) относительно исходного положения. Это объясняется тем, что масштабирование выполняется относительно заданного центра, и левая часть прямоугольника увеличивается точно так же, как и правая. Аналогичные результаты в вертикальном направлении можно получить, задавая *ScaleY* и *CenterY*. Рассмотрим пример:

```
<Rectangle Fill="#FFFF0404" Stroke="#FF000000"
    Width="96" Height="88" Canvas.Left="80" Canvas.Top="80">
  <Rectangle.RenderTransform>
    <ScaleTransform ScaleX="2" CenterX="50"/>
  </Rectangle.RenderTransform>
</Rectangle>
```

Как в результате этого изменяется прямоугольник, можно увидеть на рис. Рис. 5-5.

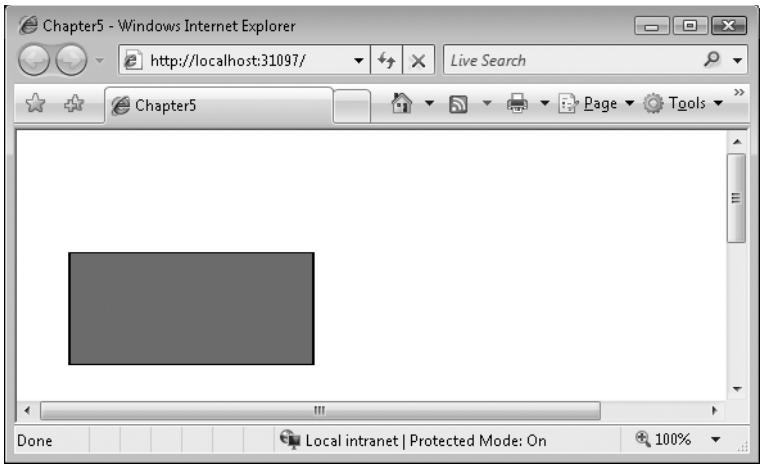

РИС. 5-5 Масштабирование относительно центра масштабирования.

Перемещение объекта с помощью *TranslateTransform*

Трансляция (*translation*) – это трансформация путем перемещения объекта на плоскости из одного места в другое. Описывается она путем задания векторов, которые определяют перемещение объекта относительно осей x и y. Для этого используются свойства *X* и *Y*. Чтобы передвинуть элемент на две единицы по горизонтали (объект будет двигаться вправо), свойству *X* задается значение 2. Чтобы переместить элемент влево, используется отрицательное значение, например, -2. Аналогично, для перемещения объекта по вертикали применяется свойство *Y*. Задание ему положительных значений обусловит перемещение объекта вниз по экрану, тогда как отрицательные значения обеспечат перемещение объекта вверх.

Рассмотрим пример трансформации трансляцией, которая обеспечивает перемещение уже знакомого нам красного прямоугольника вверх и влево через задание свойств *X* и *Y*. Эти значения образуют *вектор*, определяющий трансформацию.

```
<Rectangle Fill="#FFFF0404" Stroke="#FF000000"
        Width="96" Height="88" Canvas.Left="80" Canvas.Top="80">
  <Rectangle.RenderTransform>
    <TranslateTransform X="-50" Y="-50"/>
  </Rectangle.RenderTransform>
</Rectangle>
```

Результаты этой трансформации представлены на рис. 5-6. Прямоугольник переместился вверх и влево относительно его исходного положения (рис. 5-3).

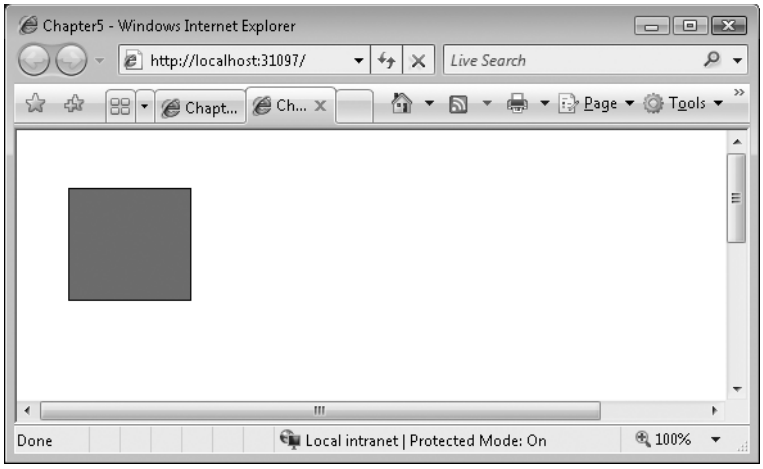

РИС. 5-6 Использование свойства *TranslateTransform*.

Наклон объекта с помощью свойства *SkewTransform*

Наклон объекта подразумевает его поступательное пропорциональное изменение вдоль одной из осей. В результате квадрат или прямоугольник превращается в параллелограмм. Этот визуальный эффект очень полезен при создании впечатления трехмерного изображения на плоскости.

Можно задавать наклон под определенным углом вдоль оси x или y и относительно определенного центра. Конечно, все это можно комбинировать и наклонять объект относительно обеих осей одновременно.

Рассмотрим XAML-код, обеспечивающий наклон прямоугольника вдоль оси x на 45 градусов:

```
<Rectangle Fill="#FFFF0404" Stroke="#FF000000"
        Width="96" Height="88" Canvas.Left="80" Canvas.Top="80">
  <Rectangle.RenderTransform>
    <SkewTransform AngleX="45"/>
  </Rectangle.RenderTransform>
</Rectangle>
```

Результат представлен на рис. 5-7.

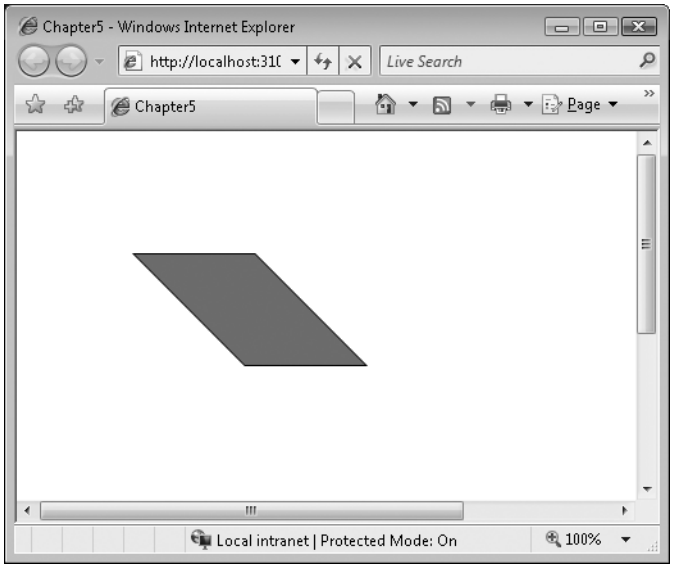

РИС. 5-7 Наклонение прямоугольника с помощью *SkewTransform*.

Имитация трехмерного изображения с помощью *SkewTransform*

Наклон используется для создания эффекта трехмерного изображения. Ниже представлен пример XAML-кода, в котором используется три прямоугольника, два из которых наклонены по оси x, и один – по оси y, что создает впечатление трехмерного изображения:

```
<Rectangle Fill="#FFFF0404" Stroke="#FF000000"
        Width="88" Height="88" Canvas.Left="80" Canvas.Top="80">
  <Rectangle.RenderTransform>
    <SkewTransform AngleX="45"/>
  </Rectangle.RenderTransform>
</Rectangle>
<Rectangle Fill="#FFFF0404" Stroke="#FF000000"
        Width="88" Height="88" Canvas.Left="80" Canvas.Top="168">
  <Rectangle.RenderTransform>
    <SkewTransform AngleX="45"/>
  </Rectangle.RenderTransform>
</Rectangle>
<Rectangle Fill="#FFFF0404" Stroke="#FF000000"
        Width="88" Height="88" Canvas.Left="80" Canvas.Top="80">
  <Rectangle.RenderTransform>
    <SkewTransform AngleY="45"/>
```

```
  </Rectangle.RenderTransform>
</Rectangle>
```

Результаты этого представлены на рис. 5-8.[1]

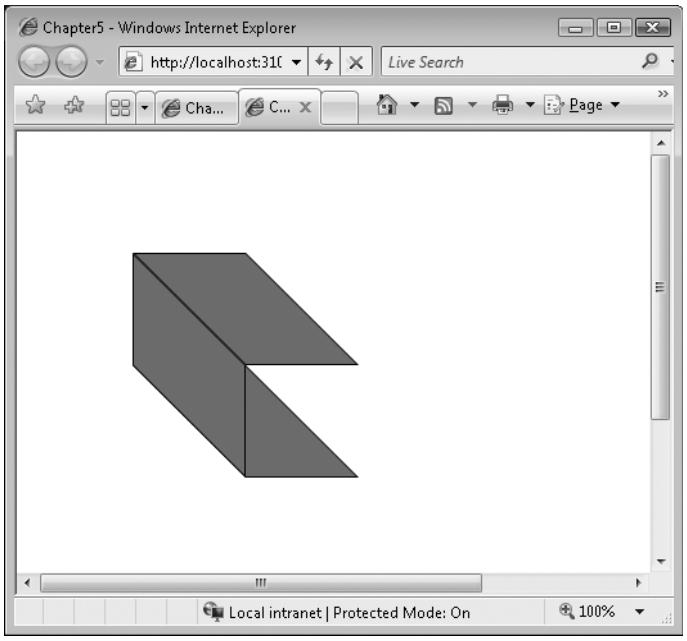

РИС. 5-8 Имитация трехмерного изображения с помощью *SkewTransform*.

Определение собственных трансформаций с помощью *MatrixTransform*

Все трансформации по сути своей осуществляются путем совокупного изменения координатного пространства объекта с помощью матрицы трансформации. Все представленные до этого в данной главе трансформаций широко известны и хорошо описаны.

Математика матриц и реализация трансформаций выходят за рамки рассмотрения данной книгой, но ради синтаксической полноты в данной главе мы рассмотрим, как их можно определять в Silverlight XAML.

Обратите внимание, что матрица, используемая в *MatrixTransform*, является *аффинной*. Это означает, что последний столбец матрицы всегда (0 0 1), и, таким образом, задаются только первые два столбца. Для этого используется свойство *Matrix* (Матрица), которое принимает строковое значение, образованное значениями первых двух столбцов, записанными через пробелы. Ниже приведен пример:

```
<Rectangle Fill="#FFFF0404" Stroke="#FF000000"
      Width="96" Height="88" Canvas.Left="80" Canvas.Top="80">
  <Rectangle.RenderTransform>
    <MatrixTransform Matrix="1 0 1 2 0 1" />
  </Rectangle.RenderTransform>
</Rectangle>
```

На рис. 5-9 продемонстрирован результат применения этой матрицы – растянутый и наклоненный прямоугольник.

[1] Корневым элементом сцены в данном примере должен быть Canvas, а не Grid, предлагаемый по умолчанию. Иначе не сработает свойство Canvas.Top и Вы не увидите нижний прямоугольник (*прим. редактора*).

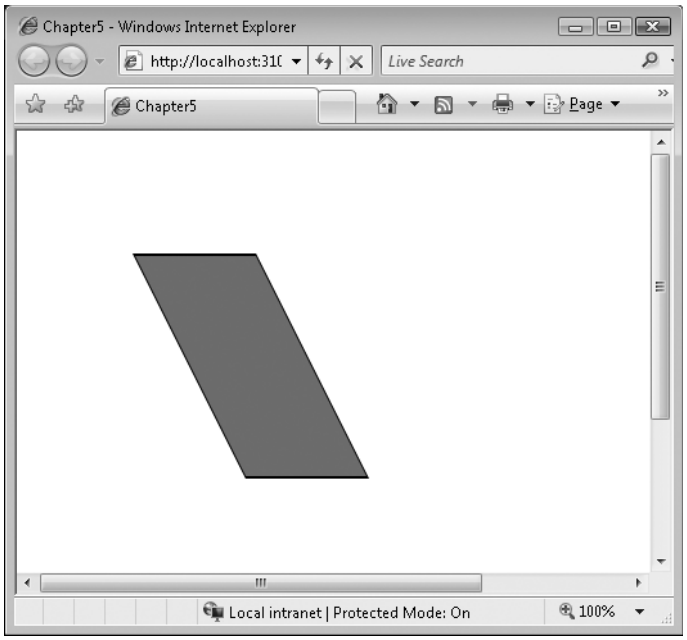

РИС. 5-9 Использование *MatrixTransform*.

Сочетание трансформаций

Как было представлено в предыдущем примере, с помощью аффинной матрицы и *MatrixTransform* можно создавать сложные трансформации. Однако не все хорошо разбираются в математике матриц. Поэтому применяется другая техника – сочетание трансформаций с помощью элемента *TransformGroup* (Группа трансформаций), который просто позволяет задавать несколько трансформаций. Объект подвергается комбинированному эффекту от всех трансформаций вместе взятых. Рассмотрим пример:

```
<Rectangle Fill="#FFFF0404" Stroke="#FF000000"
      Width="96" Height="88" Canvas.Left="80" Canvas.Top="80">
  <Rectangle.RenderTransform>
    <TransformGroup>
      <ScaleTransform ScaleX="1.2" ScaleY="1.2" />
      <SkewTransform AngleX="30" />
      <RotateTransform Angle="45" />
    </TransformGroup>
  </Rectangle.RenderTransform>
</Rectangle>
```

В этом примере комбинируются *ScaleTransform*, который обеспечивает увеличение размера фигуры вдоль обеих осей на 20%, наклон на 30 градусов по оси x и поворот на 45 градусов. Результаты этой трансформации представлены на рис. 5-10.

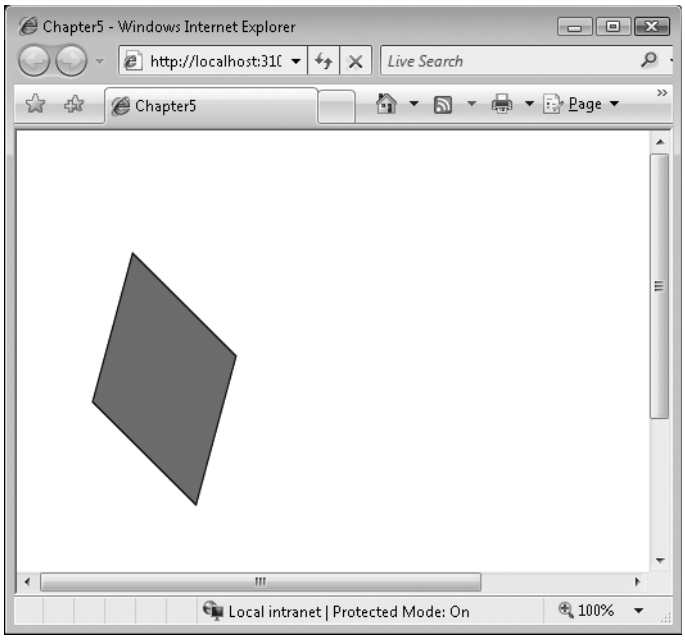

РИС. 5-10 Сочетание трансформаций с помощью *TransformGroup*.

Трехмерные эффекты с преобразованиями перспективы

Одной из наиболее захватывающих возможностей, добавленных в Silverlight 3, является преобразования перспективы. В двух словах, преобразования перспективы – это преобразования, которые могут применяться к элементам XAML для имитации их вращения в трехмерном пространстве. Следует отметить, что это не *реальное* 3D-пространство, поскольку здесь нет трехмерного моделирования с помощью сеток, шейдинга (shading), удаления скрытых линий и пр., но эта возможность позволяет моделировать 3D-эффекты с помощью XAML.

Посмотрим, как это делается:

```
<Grid x:Name="LayoutRoot" Background="White">
   <Image Source="whack.jpg">
   </Image>
</Grid>
```

В данном примере представлен XAML-код, описывающий изображение юного бейсболиста. Если выполнить этот код в браузере с поддержкой Silverlight, будет получено следующее изображение (рис. 5-11).

РИС. 5-11 Представление победного удара без перспективы.

Теперь представим, что экран является трехмерным пространством. Горизонтальная ось – это ось X, вертикальная ось – ось Y, и нормаль к поверхности – ось Z. Если требуется создать впечатление разворота изображения в трехмерном пространстве, так чтобы его левая часть «уходила вглубь» экрана, необходимо вращать изображение *относительно* оси Y. Аналогичным образом, если требуется развернуть изображение так, чтобы вглубь экрана уходила его верхняя или нижняя часть, вращение выполняется относительно оси X.

На первый взгляд, может показаться, что необходимо развернуть изображение относительно оси Z, но это ось, перпендикулярная к плоскости экрана. Таким образом, вращение относительно этой оси в трехмерном пространстве приведет лишь к изменению угла представления изображения.

Итак, рассмотрим код, обеспечивающий перспективное представление изображения путем разворота на 45 градусов относительно оси Y:

```
<Grid x:Name="LayoutRoot" Background="White">
   <Image Source="whack.jpg">
      <Image.Projection>
         <PlaneProjection RotationY="45"></PlaneProjection>
      </Image.Projection>
   </Image>
</Grid>
```

Результаты представлены на рис. 5-12.

РИС. 5-12 Вращение изображения относительно оси Y.

Аналогично, разворачивая изображение относительно оси X, можно создать перспективу, когда вглубь экрана будет уходить верхняя или нижняя часть изображения.

Рассмотрим пример:

```
<Grid x:Name="LayoutRoot" Background="White">
   <Image Source="whack.jpg" Width="640" Height="480" Canvas.Top="150" Canvas.Left="0">
      <Image.Projection>
         <PlaneProjection RotationX="45"></PlaneProjection>
      </Image.Projection>
   </Image>
</Grid>
```

Результат представлен на рис. 5-13.

РИС. 5-13 Вращение вокруг оси X.

Обратите внимание, что Silverlight рассматривает изображения как *прозрачные*. Таким образом, если развернуть картинку так, чтобы она повернулась к нам тыльной стороной, мы увидим перевернутое изображение. Если в предыдущем примере задать угол поворота равным 135 градусам, получим результат, подобный представленному на рис. 5-14.

РИС. 5-14 Использование преобразования перспективы для просмотра изображения с тыльной стороны.

С помощью класса *Storyboard* можно выполнить анимацию разворота. Подробнее о работе с элементом *Storyboard* рассказывается в данной главе позже, поэтому если код кажется несколько странным, не отчаивайтесь, со временем мы во всем разберемся.

```
<UserControl x:Class="sl3dtest.Page"
   xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
```

```
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    Width="800" Height="600">
    <UserControl.Resources>
        <Storyboard x:Name="Storyboard1">
            <DoubleAnimationUsingKeyFrames BeginTime="00:00:00"
              Storyboard.TargetName="img"
              Storyboard.TargetProperty="(UIElement.Projection).(RotationX)"
              RepeatBehavior="Forever">
                <SplineDoubleKeyFrame KeyTime="00:00:00" Value="0"/>
                <SplineDoubleKeyFrame KeyTime="00:00:02" Value="360"/>
            </DoubleAnimationUsingKeyFrames>
        </Storyboard>
    </UserControl.Resources>
    <Grid x:Name="LayoutRoot" Background="White">
        <Image x:Name="img" Source="whack.jpg" Width="640" Height="480" Canvas.Top="150" Canvas.Left="0">
            <Image.Projection>
                <PlaneProjection RotationX="0"></PlaneProjection>
            </Image.Projection>
        </Image>
    </Grid>
</UserControl>
```

Этот фрагмент XAML-кода просто обеспечивает создание *Storyboard*, выполняющегося в течение двух секунд и изменяющего значение *RotationX* от 0 до 360, и повторение этого процесса. В результате создается эффект постоянного вращения изображения вокруг оси X в трехмерном пространстве.

Анимация

Слово *анимация* буквально означает «оживить что-то». Таким образом, с помощью анимации вы можете вдохнуть жизнь в созданные вами объекты, меняя их атрибуты, такие как цвет, размер, прозрачность и другие свойства, во времени и в ответ на действия пользователя.

В XAML анимация элемента осуществляется за счет изменения одного или более его свойство во времени. Это время определяется с помощью временной шкалы. Например, чтобы элемент пересек экран за 5 секунд, необходимо было бы задать временную шкалу длительностью 5 секунд, которая обеспечила бы изменение свойства *Canvas.Left* от 0 до значения ширины экрана. В следующих разделах будут обсуждаться все доступные типы анимации и отличия анимации этих свойств с использованием ключевых кадров.

Прежде чем переходить к различным типа анимации, вы должны познакомиться с механизмами анимаций, которые включают *Trigger* (Триггер), *EventTrigger* (Триггер события) и *Storyboard* (Раскадровка). Сначала рассмотрим эти концепции, а затем уже углубимся в различные типы анимации.

Использование триггеров и триггеров событий

Анимации в Silverlight выполняются в ответ на некоторое событие, которое определяется с помощью триггера. В настоящее время Silverlight XAML поддерживает только один тип триггеров –*EventTrigger*. Каждое свойство пользовательского интерфейса (UI) имеет коллекцию *Triggers*, которая используется для определения одного или более триггеров (т.е. одного или более объектов *EventTrigger*).

Итак, первый шаг в добавлении анимации в элемент – создание его коллекции триггеров. Затем в эту коллекцию необходимо будет добавить, по крайней мере, один триггер события. Например, первый шаг при анимации прямоугольника, определение коллекции триггеров, выглядит следующим образом[1]:

```
<Rectangle x:Name="rect" Fill="Red" Canvas.Top="100"
        Canvas.Left="100" Width="100" Height="100">
```

[1] Родительским элементом для прямоугольника должен быть Canvas (*прим. редактора*).

```
  <Rectangle.Triggers>
  </Rectangle.Triggers>
</Rectangle>
```

Далее необходимо определить *EventTrigger*, который будет добавлен в эту коллекцию. В данном *EventTrigger* с помощью свойства *RoutedEvent* (Отслеживаемое событие) задается событие, в ответ на возникновение которого будет запускаться анимация. В Главе 6, «Объект Silverlight для браузера», подробно рассматриваются все события, поддерживаемые каждым из объектов в XAML, но обратите внимание, что *RoutedEvent* поддерживает только событие *Loaded* (Загружен).

Чтобы реализовать анимацию, которая будет запускаться после загрузки прямоугольника, *EventTrigger* необходимо описать следующим образом:

```
<EventTrigger RoutedEvent="Rectangle.Loaded">
</EventTrigger>
```

Вот фрагмент XAML-кода, запускающий эту анимацию:

```
<Rectangle x:Name="rect" Fill="Red" Canvas.Top="100"
        Canvas.Left="100" Width="100" Height="100">
  <Rectangle.Triggers>
    <EventTrigger RoutedEvent="Rectangle.Loaded">
    </EventTrigger>
  </Rectangle.Triggers>
</Rectangle>
```

Следующий шаг – создать анимацию, которая будет использоваться. Анимации размещаются в объектах *Storyboard*, которым посвящен следующий раздел.

Использование *BeginStoryboard* и *Storyboard*

BeginStoryboard (Начать раскадровку) – это триггерное действие, содержащее объект *Storyboard*. В объектах *Storyboard* находятся описания анимации. При описании анимации эти объекты просто встраиваются в определение *EventTrigger*. Вот как можно сделать это, используя рассматриваемый выше пример с прямоугольником:

```
<Rectangle x:Name="rect" Fill="Red" Canvas.Top="100"
        Canvas.Left="100" Width="100" Height="100">
  <Rectangle.Triggers>
    <EventTrigger RoutedEvent="Rectangle.Loaded">
      <BeginStoryboard>
        <Storyboard>
        </Storyboard>
      </BeginStoryboard>
    </EventTrigger>
  </Rectangle.Triggers>
</Rectangle>
```

Определение параметров анимации

Теперь, когда среда анимации задана, можно приступать к описанию самой анимации. В основе анимации лежит изменение свойства во времени. Возможна анимация трех разных типов свойств:

- Свойства типа *double* анимируются с помощью *DoubleAnimation* или *DoubleAnimationUsingKeyFrames* (Анимация свойств типа Double с использованием ключевых кадров). Этот метод используется для анимации свойств со значением типа *double*, например, таких размеров, как *Canvas.Left*, или таких визуальных атрибутов, как *Opacity*.
- Свойства типа точка анимируются с помощью *PointAnimiation* (Анимация точки) или *PointAnimationUsingKeyFrames* (Анимация точки с использованием ключевых кадров). Этот метод используется для анимации свойств со значениями, определенными как *точка*. Например, сегменты прямых или кривые описывают с помощью точек.

- Свойства типа цвет анимируются с помощью *ColorAnimation* или *ColorAnimationUsingKeyFrames* (Цветовая анимация с использованием ключевых кадров). Этот метод используется для анимации свойств, содержащих значения *цвета*, например, фона или обводки элемента.

Анимация каждого из этих типов свойств выполняется от значения, заданного атрибутом *From* (От) (или его текущего значения, если этот атрибут не задан), до значения, заданного атрибутом *To* (До), или на значение, заданное атрибутом *By* (На).

Задание объекта анимации

Объект, к которому должна применяться анимация, задается свойством *Storyboard.TargetName* (Имя цели) данного типа анимации. Значением *Storyboard.TargetName* является имя объекта, которое в описании объекта задано свойством *x:Name* (Имя). Кроме того, с помощью *Storyboard.TargetProperty* (Целевое свойство) задается анимируемое свойство. Заметьте, что составное или присоединенное свойство (такое как *Canvas.Left*) заключается в круглые скобки. Итак, например, чтобы задать анимацию свойства типа *Double* для свойства *Canvas.Left* прямоугольника *rect*, необходимо написать следующий XAML:

```
<DoubleAnimation Storyboard.TargetName="rect"
Storyboard.TargetProperty="(Canvas.Left)" />
```

Задание продолжительности

Для определения времени перехода анимируемых свойств от одного значения к другому используется свойство *Duration* (Продолжительность). Оно определяется в формате ЧЧ:ММ:СС, где продолжительность 5 секунд записывается 00:00:05, или сокращенно – 0:0:5. Рассмотрим пример:

```
<DoubleAnimation Storyboard.TargetName="rect"
Storyboard.TargetProperty="(Canvas.Left)" Duration="0:0:5" />
```

Задание момента начала

Если вы не хотите, чтобы анимация начиналась сразу после загрузки, можно ввести задержку с помощью свойства *BeginTime* (Момент начала). Для него используется такой же синтаксис, как и для *Duration*.

```
<DoubleAnimation Storyboard.TargetName="rect"
Storyboard.TargetProperty="(Canvas.Left)" BeginTime="0:0:5" />
```

Использование свойства *SpeedRatio*

Можно настроить поведение анимации, умножая ее продолжительность на коэффициент скорости. Для этого используется свойство *SpeedRatio* (Коэффициент скорости). Например, в предыдущем случае была задана продолжительность 5 секунд. Задав *SpeedRatio* равным 2, можно изменить скорость воспроизведения, и анимация будет длиться 10 секунд. Или можно ускорить анимацию до 1 секунды, задавая *SpeedRatio* равным 0,2.

```
<DoubleAnimation Storyboard.TargetName="rect"
Storyboard.TargetProperty="(Canvas.Left)" SpeedRatio="2" Duration="0:0:5" />
```

Использование свойства *AutoReverse*

Анимация Silverlight обеспечивает возможность возвращения свойств, измененных в ходе анимации, к исходным значениям. Например, если значение типа *double* изменяется от 0 до 500 за определенный промежуток времени, свойство *AutoReverse* (Автоматическая перемотка) заставит анимацию выполниться в обратном направлении, от 500 к 0.

Обратите внимание, что если задана продолжительность анимации 5 секунд, и свойству *AutoReverse* задано значение *true*, продолжительность всей анимации будет 10 секунд. Далее представлен пример XAML со свойством *AutoReverse*:

```
<DoubleAnimation Storyboard.TargetName="rect"
   Storyboard.TargetProperty="(Canvas.Left)" AutoReverse="True"
   Duration="0:0:5" />
```

Задание свойства *RepeatBehavior*

Существует ряд опций, посредством которых можно управлять поведением анимации по завершении ее выполнения. Задаются они с помощью свойства *RepeatBehavior* (Поведение повтора). Это свойство может принимать три типа значений:

- Время в секундах. После прохождения этого промежутка времени на временной шкале, анимация будет запускаться повторно.
- Значение *Forever* (Постоянно) определяет постоянное повторение.
- Определенное количество повторений. Для этого задается число, определяющее число повторений, за которым следует символ *x*. Например, если необходимо повторить анимацию три раза, задается значение *3x*.

Далее представлен полный XAML-код для анимации перемещения прямоугольника из точки 100 в точку 500 и назад в 100 по оси x с последующим троекратным повторением:

```
<Rectangle x:Name="rect" Fill="Red"
    Canvas.Top="100" Canvas.Left="100" Width="100" Height="100">
  <Rectangle.Triggers>
    <EventTrigger RoutedEvent="Rectangle.Loaded">
      <BeginStoryboard>
        <Storyboard>
          <DoubleAnimation RepeatBehavior="3x"
                    Storyboard.TargetName="rect"
                    Storyboard.TargetProperty="(Canvas.Left)"
                    To="500" Duration="0:0:5"
                    AutoReverse="True" />
        </Storyboard>
      </BeginStoryboard>
    </EventTrigger>
  </Rectangle.Triggers>
</Rectangle>
```

В следующих разделах рассмотрим каждый из этих типов анимации немного более подробно. Сначала обратимся к атрибутам, необходимым для анимации каждого из типов, а затем перейдем к использованию анимации по ключевым кадрам.

Анимация значения с помощью *DoubleAnimation*

Объект *DoubleAnimation* позволяет задавать, как значения типа *double* изменяется со временем. Анимация вычисляется как линейная интерполяция между значениями свойства в разные моменты времени.

При анимации свойств типа *double* задается значение в начале анимации, с помощью свойства *From*, и его абсолютное конечное значение, с помощью свойства *To*, или относительное конечное значение, посредством свойства *By*.

Например, чтобы изменить свойство *Canvas.Left* элемента от 100 (расположение рядом с левым краем экрана) до 500, можно задать свойству *From* значение 100 и 500 свойству *To* или 400 свойству *By*. Обратите внимание, что в случае задания обоих свойств, *To* имеет преимущество над *By*. Также если прямоугольник уже находится в определяемой *From* позиции, это свойство задавать не надо.

Предыдущий пример XAML-кода реализует это поведение. Прямоугольник располагается в позиции с *Canvas.Left* равным 100. *DoubleAnimation* определяет для *To* значение 500. Следовательно, анимация обеспечит изменение значения от 100 до 500, что обусловит перемещение прямоугольника по экрану вправо.

Анимация цвета с помощью *ColorAnimation*

Поведение *ColorAnimation* очень близко поведению *DoubleAnimation*. Этот тип используется для изменения значения *color* элемента во времени. Анимация вычисляется как линейная интерполяция между значениями *color* в заданный промежуток времени.

При анимации цвета значение в начале анимации задается свойством *From*. Если это свойство не задано, за исходный принимается текущий цвет. Атрибут *To* определяет конечный цвет. Также можно задать атрибут *By*, тогда конечный цвет будет получен путем суммирования значения цвета, заданного в *From* (или исходного цвета), и цвета, заданного *By*.

Анимация цветовых свойств не означает анимацию содержимого свойства, потому что содержимым свойства обычно является кисть, а не цвет. Поэтому, например, если необходимо анимировать цвет заливки прямоугольника, целевым свойством анимации является не его свойство *Fill*, а свойство *Color* кисти *SolidBrush*, которая используется для выполнения заливки.

Далее представлен пример анимации цвета прямоугольника. Цвет меняется от черного непрозрачного к черному абсолютно прозрачному в течение пяти секунд:

```
<Rectangle x:Name="rect" Canvas.Top="100" Canvas.Left="100"
        Width="100" Height="100" Fill="Black">
  <Rectangle.Triggers>
    <EventTrigger RoutedEvent="Rectangle.Loaded">
      <BeginStoryboard>
        <Storyboard>
          <ColorAnimation Storyboard.TargetName="rect"
                    Storyboard.TargetProperty=
                      "(Shape.Fill).(SolidColorBrush.Color)"
                    To="#00000000" Duration="0:0:5" />
        </Storyboard>
      </BeginStoryboard>
    </EventTrigger>
  </Rectangle.Triggers>
</Rectangle>
```

Как видите, в этом фрагменте XAML целевым свойством является свойство *Color* кисти *SolidColorBrush*, закрашивающей фигуру. Это типовой синтаксис XAML, используемый при работе с такими составными свойствами.

Анимация точки с помощью *PointAnimation*

Для изменения во времени значения, определенного как точка, используется тип *PointAnimation*. Анимация вычисляется как линейная интерполяция между значениями в заданный промежуток времени.

Аналогично тому, как это делалось для анимации цвета и объектов типа *Double*, исходное значение задается с помощью свойства *From* и конечное значение задается как относительное (используя *By*) или как абсолютное (используя *To*). Далее представлен пример анимации конечной точки кривой Безье:

```
<Path Stroke="Black" >
  <Path.Data>
    <PathGeometry>
      <PathFigure StartPoint="100,100">
        <QuadraticBezierSegment x:Name="seg"
                Point1="200,0" Point2="300,100"  />
```

```
      </PathFigure>
    </PathGeometry>
  </Path.Data>
  <Path.Triggers>
    <EventTrigger RoutedEvent="Path.Loaded">
      <BeginStoryboard>
        <Storyboard>
          <PointAnimation Storyboard.TargetName="seg"
                  Storyboard.TargetProperty="Point2"
                  From="300,100" To="300,600" Duration="0:0:5" />
        </Storyboard>
      </BeginStoryboard>
    </EventTrigger>
  </Path.Triggers>
</Path>
```

В данном случае кривая Безье описывается начальной точкой (100,100), конечной точкой (300,100) и опорной точкой (200,0). Задано, что анимация запускается после загрузки контура и обеспечивает перемещение конечной точки кривой (*Point2*) от точки (300,100) в точку (300,600) в течение пяти секунд.

Использование ключевых кадров

Все рассматриваемые до сих пор типы анимации: *ColorAnimation*, *DoubleAnimation* и *PointAnimation*, – выполнялись путем изменения заданного свойства во времени с использованием линейной интерполяции. Например, при изменении значения типа *double* от 100 до 500 за 5 секунд, его приращение составляет 80 единиц в секунду.

Каждый из этих переходов может быть определен рядом промежуточных этапов, называемых *ключевыми кадрами*(*key frames*). Для изменения линейного поведения анимации от начального свойства к конечному используется один или более ключевых кадров. Затем задается, какой стиль анимации должен быть применен между различными точками.

Ключевые кадры определяются с помощью *опорных моментов* (*key times*). Это моменты, задаваемые относительно начала анимации и определяющие время окончания ключевого кадра. Итак, представим, что необходимо создать анимацию продолжительностью девять секунд с тремя ключевыми кадрами, расположенными через равные промежутки времени друг от друга. Первый ключевой кадр можно задать в конце третьей секунды (0:0:3), второй – в конце шестой (0:0:6), и третий – в конце девятой (0:0:9). Вы не задаете *продолжительность* опорного момента, вы задаете время окончания каждого ключевого кадра.

В качестве еще одного примера возьмем анимацию объектов типа *Double*, с разделением промежутка 100 – 500 на две равные части. Анимация должна двигаться очень быстро первую половину пути и очень медленно вторую половину. Все перемещение должно занять шесть секунд. Поскольку средней точкой между 100 и 500 является 300, начало ключевого кадра определяем в точке 300. Указываем, что перемещение из начальной точкой в среднюю точку должно занять 1 секунду, используя опорный момент 0:0:1. Затем задаем продолжительность перемещения из средней точки в конечную точку равным 5 секундам, используя второй опорный момент, 0:0:6. Элемент будет мгновенно перескакивать из начальной в среднюю точку и затем медленно проходить остальную часть пути.

В предыдущем примере оба сегмента анимации используют линейную интерполяцию. Для обеспечения дополнительной гибкости существует еще два типа ключевых кадров: ключевой кадр дискретной анимации, который мгновенно «перебрасывает» значение от одного значения к другому, и ключевой кадр сглаженной анимации, который обеспечивает плавное изменение значения, используя для определения интерполяции кривую второго порядка. (В следующих разделах будет рассмотрено описание анимации с использованием ключевых кадров для объектов типа *Double*. Аналогичные принципы применяются для анимации *Point* и *Color*.)

В наименованиях типов анимации с использованием ключевых кадров используется постфикс *UsingKeyFrames* (с использованием ключевых кадров). То есть для описания анимации объектов типа *Double* с ключевыми кадрами будет использоваться тип *DoubleAnimationUsingKeyFrames*, для которого будут заданы цель и свойство (точно так же, как это делалось для *DoubleAnimation*). *DoubleAnimationUsingKeyFrames* включает описания ключевых кадров. (И, как упоминалось ранее, то же самое касается *PointAnimationUsingKeyFrames* и *ColorAnimationUsingKeyFrames*.)

Использование линейных ключевых кадров

Стандартным методом анимации изменения значений свойства является линейная интерполяция, при которой общая величина изменения равномерно распределяется по времени, отведенному на изменение. Другой способ – определение линейных шагов между кадрами с помощью *LinearKeyFrame* (Линейный ключевой кадр), в котором используется все та же линейная интерполяция, но между ключевыми кадрами, поэтому можно создавать эффекты ускорения/замедления.

Рассмотрим следующий пример анимации. В нем используется *DoubleAnimationUsingKeyFrames*. В этом случае определяются два ключевых кадра. Один кадр описывает линейную интерполяцию для изменения *Canvas.Left* от 0 до 300 за 1 секунду, второй – линейную интерполяцию для изменения *Canvas.Left* от 300 до 600 за 8 секунд. В результате создается эффект быстрого перемещения прямоугольника в первой половине пути и медленного движения во второй. Аналогичные принципы применяются для *LinearPointKeyFrame* и *LinearColorKeyFrame*.

```
<Rectangle Fill="#FFFF0000" Stroke="#FF000000"
        Width="40" Height="40" Canvas.Top="40" x:Name="rect">
  <Rectangle.Triggers>
    <EventTrigger RoutedEvent="Rectangle.Loaded">
      <BeginStoryboard>
        <Storyboard>
          <DoubleAnimationUsingKeyFrames
              Storyboard.TargetName="rect"
              Storyboard.TargetProperty="(Canvas.Left)" >
            <LinearDoubleKeyFrame KeyTime="0:0:1" Value="300" />
            <LinearDoubleKeyFrame KeyTime="0:0:9" Value="600" />
          </DoubleAnimationUsingKeyFrames>
        </Storyboard>
      </BeginStoryboard>
    </EventTrigger>
  </Rectangle.Triggers>
</Rectangle>
```

Использование ключевых кадров дискретной анимации

Если необходимо изменить свойство от одного значения к другому, но без применения линейной интерполяции, можно использовать ключевой кадр дискретной анимации. Он заставит объект «перескочить» на заданное значение в заданное время ключевого кадра. Далее показан тот же пример, что был представлен выше, но с использованием ключевого кадра дискретной анимации. За 1 секунду анимации прямоугольник перепрыгнет полпути на экране. Остальные 8 секунд анимации он будет проходить оставшийся отрезок пути.

```
<Rectangle Fill="#FFFF0000" Stroke="#FF000000"
    Width="40" Height="40" Canvas.Top="40" x:Name="rect">
  <Rectangle.Triggers>
    <EventTrigger RoutedEvent="Rectangle.Loaded">
      <BeginStoryboard>
        <Storyboard>
          <DoubleAnimationUsingKeyFrames
              Storyboard.TargetName="rect"
              Storyboard.TargetProperty="(Canvas.Left)" >
            <DiscreteDoubleKeyFrame KeyTime="0:0:1" Value="300" />
            <DiscreteDoubleKeyFrame KeyTime="0:0:9" Value="600" />
          </DoubleAnimationUsingKeyFrames>
        </Storyboard>
```

```
      </BeginStoryboard>
    </EventTrigger>
  </Rectangle.Triggers>
</Rectangle>
```

Аналогичные принципы применяются для *DiscretePointKeyFrame* и *DiscreteColorKeyFrame*.

Использование ключевых кадров сглаженной анимации

Для изменения свойства от одного значения к другому с использованием нелинейно изменяемого значения, обеспечивающим ускорение и замедление, применяется ключевой кадр сглаженной анимации. Для этого задается квадратическая кривая Безье. Тогда скорость изменения свойства от одного значения к другому определяется параллельной проекцией этой кривой.

Чтобы представить это, возьмем наглядный пример: солнце стоит прямо над головой, вы посылаете футбольный мяч в дальнюю часть поля. Проследите за тенью мяча. Пока он летит вверх, кажется, что тень движется с ускорением. Когда он достигает верхней точки, тень замедляет ход. Когда мяч начинает падать, скорость тени опять начинает возрастать до тех пор, пока мяч не упадет на землю.

Теперь представьте, что ваша анимация – это тень мяча, а плавная линия – это траектория мяча. Траектория, плавная линия или сплайн, описывается с помощью *KeySpline* (Опорная сглаженная линия). *KeySpline* определяет опорные точки квадратической кривой Безье. Кривая нормализована, так что первая точка кривой находится в 0, вторая – в 1. Для описания параболической дуги, именно по такой траектории падает мяч, *KeySpline* будет включать два нормализованных значения через запятую.

Описать кривую, по которой двигается мяч, можно с помощью сглаженной линии, используя *KeySpline*, например 0.3,0 0.6,1. Здесь задается первая (0.3,0) и вторая (0.6,1) точки кривой. В результате будет создан эффект быстрого ускорения анимации в первой трети пути, затем перемещение будет происходить с постоянной скоростью до прохождения примерно двух третей траектории мяча, и затем оставшуюся часть пути движение будет замедляться, имитируя падения мяча на землю.

Далее представлен пример использования *KeySpline* для описания сглаженной кривой при моделировании такого движения с помощью анимации *DoubleAnimationUsingKeyFrames*:

```
<Ellipse Fill="#FF444444" Stroke="#FF444444"
      Width="40" Height="40" Canvas.Top="40" x:Name="ball">
  <Ellipse.Triggers>
    <EventTrigger RoutedEvent="Ellipse.Loaded">
      <BeginStoryboard>
        <Storyboard>
          <DoubleAnimationUsingKeyFrames
              Storyboard.TargetName="ball"
              Storyboard.TargetProperty="(Canvas.Left)" >
            <SplineDoubleKeyFrame KeyTime="0:0:5"
                KeySpline="0.3,0 0.6,1" Value="600" />
          </DoubleAnimationUsingKeyFrames>
        </Storyboard>
      </BeginStoryboard>
    </EventTrigger>
  </Ellipse.Triggers>
</Ellipse>
```

В этом примере выполняется анимация эллипса. Его перемещение по экрану моделирует движение тени мяча, если смотреть на нее сверху.

Анимация и Expression Blend

Анимацию можно создавать графически в Expression Blend. Он формирует за вас XAML-код, применяя различные типы анимации автоматически.

В Expression Blend выберите в меню Window в разделе Workspaces пункт Animation (Рабочая область - Анимация). Этот режим содержит инструменты для графического проектирования временных шкал. При редактировании свойств, которые должны меняться, используя визуальный редактор, будет формироваться XAML-код анимации. Это можно увидеть на рис. 5-15.

Внизу экрана представлена панель Objects And Timeline. В ней можно добавить временную шкалу и затем визуально вводить ключевые кадры. Чтобы добавить новую временную шкалу, щелкните кнопку со знаком плюс (+) на панели Objects And Timeline (рис. 5-16).

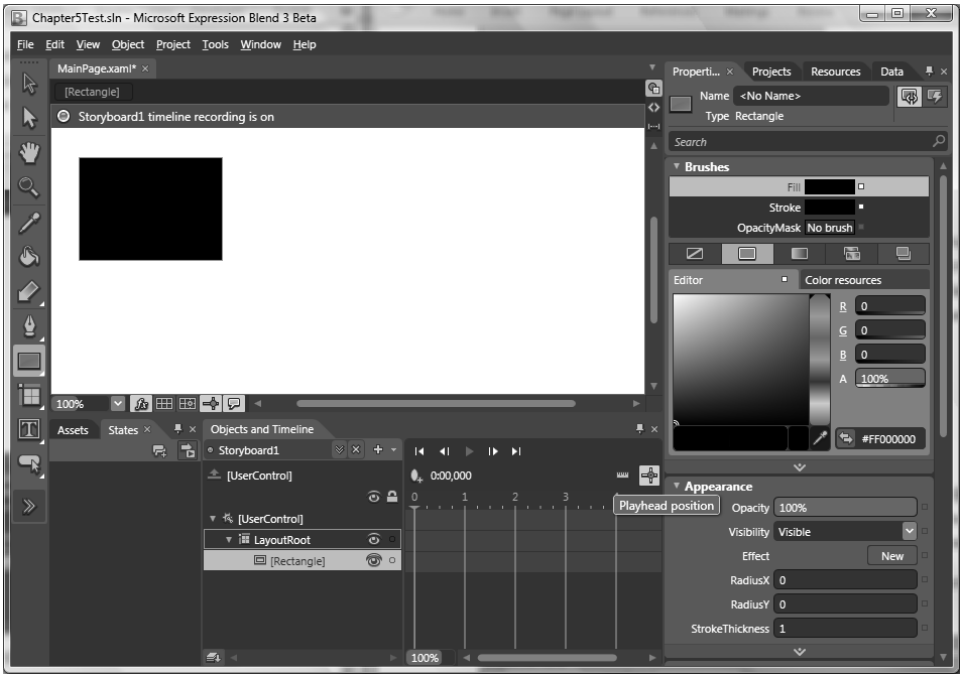

РИС. 5-15 Expression Blend в режиме Animation Workspace.

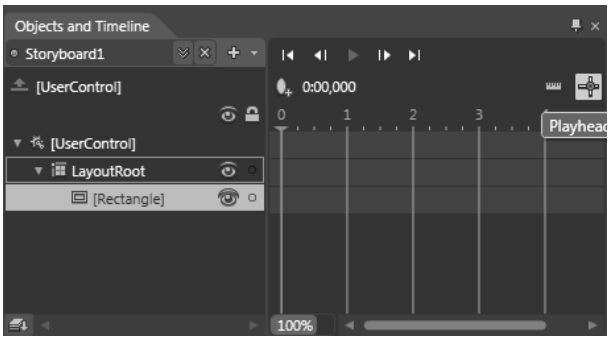

РИС. 5-16 Добавление новой временной шкалы.

Если щелкнуть кнопку со знаком плюс, откроется диалоговое окно Create Storyboard Resource (Создание раскадровки как ресурса), в котором будет предложено ввести имя создаваемой раскадровки, как показано на рис. 5-17. В данном случае я изменил предлагаемое по умолчанию имя Storyboard1 на Timeline1.

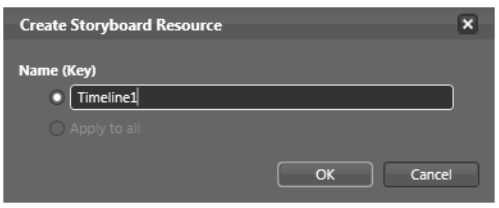

РИС. 5-17 Создание раскадровки.

При использовании Expression Blend для сайта для Silverlight 1[1] анимацию можно создавать на уровне *Canvas* или как *Resource* (Ресурс). В первом случае анимации запускаются в ответ на триггеры, располагающиеся на холсте. Далее представлен пример XAML, созданного Expression Blend из диалогового окна Create Storyboard Resource, в котом пользователь задал создание анимации не как *Resource*:

```
<Canvas.Triggers>
  <EventTrigger RoutedEvent="Canvas.Loaded">
    <BeginStoryboard>
      <Storyboard x:Name="Timeline1"/>
    </BeginStoryboard>
  </EventTrigger>
</Canvas.Triggers>
```

Представление Objects And Timeline изменится, в нем будет отображена только что созданная временная шкала. Это можно увидеть на рис. 5-18.

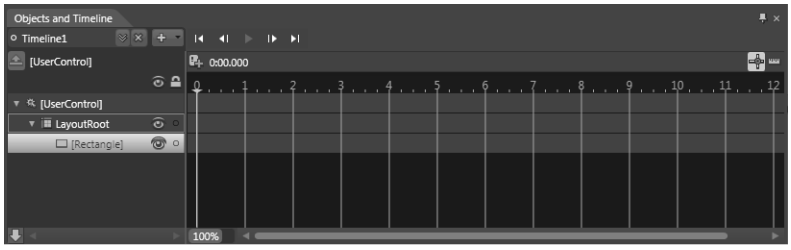

РИС. 5-18 Панель Objects And Timeline с новой временной шкалой.

На рис. 5-18 вертикальная линия на нулевой отметке времени обозначает текущее время. (В Expression Blend эта линия будет желтой.) Чтобы добавить ключевой кадр, просто перетащите эту линию на тот момент, куда необходимо вставить ключевой кадр, и щелкните кнопку Record Keyframe (Записать ключевой кадр). Эта кнопка находится сразу над временной шкалой слева от 0:00:000 на рис. 5-18.

Перетащите линию на четырехсекундную отметку и добавьте ключевой кадр. Добавленный ключевой кадр будет отображаться как небольшой овал на временной шкале, как показано на рис. 5-19.

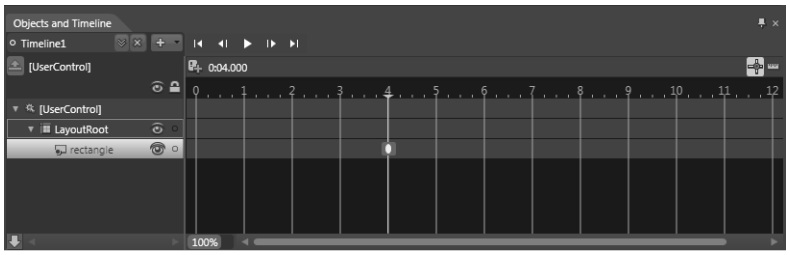

РИС. 5-19 Добавление ключевого кадра.

Установив временную шакалу на отметку четыре секунды и добавив ключевой кадр, можно перейти к редактированию цвета, местоположения, прозрачности или формы прямоугольника. Expression Blend сам проведет соответствующие вычисления для облегчения создания анимации. В качестве примера на рис. 5-20 показан тот же прямоугольник, что мы видели на рис. 5-15, но с измененными заливкой и размерами.

[1] В Expression Blend 3 возможность создания сайта для Silverlight 1 отсутствует. Это можно сделать в более ранних версиях продукта (*прим. редактора*).

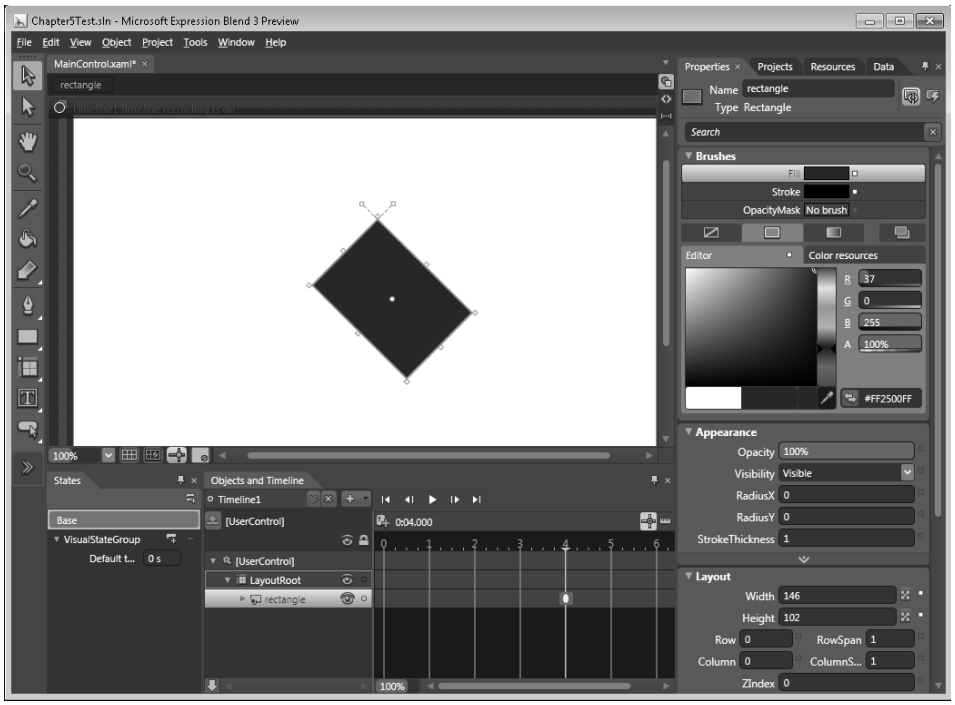

РИС. 5-20 Задание изменений в ключевом кадре.

Вероятно, вы заметили, что если перемещать индикатор временной шкалы, можно предварительно просмотреть полученную анимацию и увидеть, как она выглядит в каждый отдельно взятый момент времени. На рис. 5-21 показано, как наш прямоугольник выглядит на второй секунде опорного времени. Для этого мы просто перетащили желтую вертикальную линию на отметку две секунды.

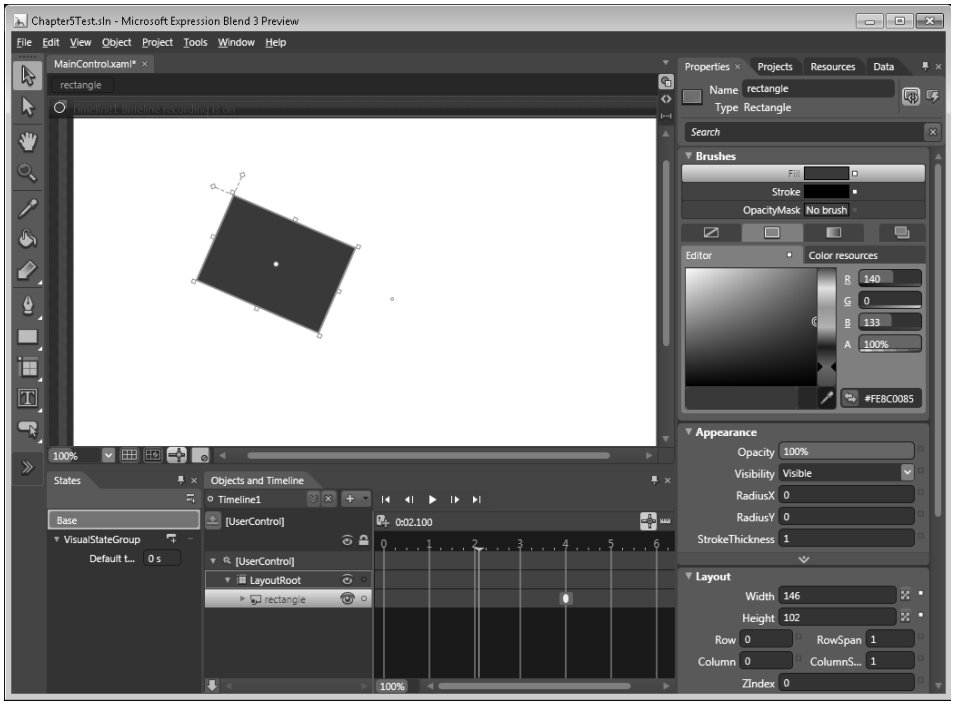

РИС. 5-21 Предварительный просмотр анимации.

Ниже представлен полный XAML-код, сформированный Expression Blend при создании анимации графическими средствами для сайта для Silverlight 1:

```
<Canvas
  xmlns="http://schemas.microsoft.com/client/2007"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
  Width="640" Height="480"
  Background="White"
  x:Name="Page">
```

```
<Canvas.Triggers>
  <EventTrigger RoutedEvent="Canvas.Loaded">
    <BeginStoryboard>
      <Storyboard x:Name="Timeline1">
        <DoubleAnimationUsingKeyFrames BeginTime="00:00:00"
          Storyboard.TargetName="rect"
          Storyboard.TargetProperty="(UIElement.RenderTransform).
                (TransformGroup.Children)[3].(TranslateTransform.X)">

          <SplineDoubleKeyFrame KeyTime="00:00:04" Value="141"/>

        </DoubleAnimationUsingKeyFrames>

        <DoubleAnimationUsingKeyFrames BeginTime="00:00:00"
          Storyboard.TargetName="rect"
          Storyboard.TargetProperty="(UIElement.RenderTransform).
                (TransformGroup.Children)[3].(TranslateTransform.Y)">
          <SplineDoubleKeyFrame KeyTime="00:00:04" Value="163"/>
        </DoubleAnimationUsingKeyFrames>

        <DoubleAnimationUsingKeyFrames BeginTime="00:00:00"
          Storyboard.TargetName="rect"
          Storyboard.TargetProperty="(UIElement.RenderTransform).
                (TransformGroup.Children)[2].(RotateTransform.Angle)">
          <SplineDoubleKeyFrame KeyTime="00:00:04" Value="35.107"/>
        </DoubleAnimationUsingKeyFrames>

        <ColorAnimationUsingKeyFrames BeginTime="00:00:00"
           Storyboard.TargetName="rect"
           Storyboard.TargetProperty="(Shape.Fill).(SolidColorBrush.Color)">
          <SplineColorKeyFrame KeyTime="00:00:04" Value="#FF9D0B0B"/>
        </ColorAnimationUsingKeyFrames>

        <DoubleAnimationUsingKeyFrames BeginTime="00:00:00" Storyboard.TargetName="rect"
          Storyboard.TargetProperty="(UIElement.RenderTransform).
                (TransformGroup.Children)[0].(ScaleTransform.ScaleX)">
          <SplineDoubleKeyFrame KeyTime="00:00:04" Value="1.7"/>
        </DoubleAnimationUsingKeyFrames>

        <DoubleAnimationUsingKeyFrames BeginTime="00:00:00"
          Storyboard.TargetName="rect"
          Storyboard.TargetProperty="(UIElement.RenderTransform).
                (TransformGroup.Children)[0].(ScaleTransform.ScaleY)">
          <SplineDoubleKeyFrame KeyTime="00:00:04" Value="1.549"/>
        </DoubleAnimationUsingKeyFrames>
      </Storyboard>
    </BeginStoryboard>
  </EventTrigger>
</Canvas.Triggers>
<Rectangle Width="87" Height="69" Fill="Red" Stroke="#FF000000"
        Canvas.Top="41" RenderTransformOrigin="0.5,0.5" x:Name="rect">
  <Rectangle.RenderTransform>
    <TransformGroup>
      <ScaleTransform ScaleX="1" ScaleY="1"/>
      <SkewTransform AngleX="0" AngleY="0"/>
      <RotateTransform Angle="0"/>
      <TranslateTransform X="0" Y="0"/>
    </TransformGroup>
  </Rectangle.RenderTransform>
</Rectangle>
</Canvas>
```

Замедление анимации

Функции замедления позволяют создавать и использовать различные специальные эффекты анимации, включая эффекты подпрыгивания или «пружины». Silverlight 3 поставляется с рядом встроенных функций замедления анимации в пространстве имен *System.Windows.Media.Animation*.

Использование функций замедления анимации существенно упрощает анимацию объектов и придает им реалистичное поведение, поскольку в этом случае разработчику не приходится вдаваться в физику процесса.

Например, если необходимо обеспечить реалистичное «подпрыгивание», можно либо самостоятельно провести расчеты, описывающие физику такого поведения, и написать соответствующую программу, либо использовать встроенную функцию анимации подпрыгивания посредством дочернего элемента *EasingFunction* (функция замедления) тега анимации.

Рассмотрим этот сценарий. В Silverlight для анимации перемещения эллипса сверху вниз по экрану используется объект <*Storyboard*>, дочерний элемент которого <*DoubleAnimation*> (Анимация свойств типа Double) описывает поведение свойства *Top* (Верх) эллипса. Чтобы ввести замедление, просто добавляем описание функции замедления, как показано в данном примере:

```
<Canvas x:Name="LayoutRoot" Background="White">
  <Canvas.Resources>
    <Storyboard x:Name="bounce">
      <DoubleAnimation From="0" To="300" Duration="0:0:10"
        Storyboard.TargetName="myCircle"
        Storyboard.TargetProperty="(Canvas.Top)">
        <DoubleAnimation.EasingFunction>
          <BounceEase Bounces="10" EasingMode="EaseOut" Bounciness="2"></BounceEase>
        </DoubleAnimation.EasingFunction>
      </DoubleAnimation>
    </Storyboard>
  </Canvas.Resources>
  <Ellipse x:Name="myCircle" Width="40" Height="40" Fill="Red"
     Canvas.Top="0" Canvas.Left="50"></Ellipse>
</Canvas>
```

Определение *EasingFunction* (названной *BounceEase*) включает используемый тип замедления. Каждый тип имеет разные параметры для описания замедления. Итак, например, для моделирования подпрыгивания упавшего объекта, просто задаются начальное и конечное положение его вершины (свойство *Top* из меняется от 0 до 300) и используется замедление для описания поведения подпрыгивания. В данном случае задано, что объект подпрыгнет 10 раз в *конце* анимации (поскольку *EasingMode* задано значение *EaseOut*).

Заметьте, существует три режима замедления: *EaseIn*, при котором функция замедления применяется в *конце* анимации; *EaseOut*, при котором она применяется в *начале* анимации; и *EaseInOut*, при котором имитация заданного движения происходит как в начале, так и в конце анимации (примерно половину пути применяется режим *EaseIn* и затем *EaseOut*).

Итак, представим эффект подпрыгивания, описанный ранее с помощью *BounceEase*, когда значение изменяется от 0 до 100. В этом случае при использовании различных режимов замедления происходит следующее:

- *EaseIn* – значение изменяется от 0 до 100. При этом функция обеспечит колебания значения в области 0 заданное количество раз, что создаст эффект подпрыгивания объекта вокруг начальной точки перемещения.
- *EaseOut* – значение изменяется от 0 до 100, но функция, прежде чем объект достигнет конечной точки перемещения, обеспечивает колебания значения в области 100 заданное количество раз, что создаст эффект подпрыгивания объекта вокруг конечной точки.
- *EaseInOut* обеспечивает причудливое сочетание предыдущих двух режимов, когда значение «колеблется» как в начале, так и в конце заданного диапазона перемещения.

В предыдущем примере использовался режим *EaseOut*, поскольку он обеспечивает более естественный эффект «подпрыгивания» мяча при соударении с твердой поверхностью.

Встроенные функции замедления располагаются в пространстве имен *System.Windows.Media.Animation*. Приведенные далее описания основываются на режиме *EaseIn*, но на их базе можно сделать выводы об эффектах, создаваемых в случае применения режимов *EaseOut* и *EaseInOut*. Зачастую разница в поведении этих функций настолько мала, что для лучшего понимания необходимо просто поэкспериментировать с ними самостоятельно.

Существуют такие встроенные функции замедления:

- *BackEase* Обеспечивает небольшой ход объекта *назад* в начале движения. Это немного напоминает то, как машина ведет себя на холме, когда она немного откатывается назад, прежде чем начать двигаться вперед.
- *BounceEase* Как было показано в предыдущем примере, эта функция создает эффект подпрыгивания.
- *CircleEase* Обеспечивает изменение ускорения анимации соответственно тригонометрической функции, с постепенным нарастанием ускорения.
- *CubicEase* Подобно *CircleEase*, но изменение ускорения осуществляется по гиперболе (кубическое уравнение времени), что обусловливает более медленное изменение ускорения в начале и более быстрое его изменение в конце движения.
- *ElasticEase* Подобно *BounceEase* в том, что обеспечивает осцилляцию значения вокруг заданной точки.
- *ExponentialEase* Подобно *CircleEase* и *CubicEase* обеспечивает экспоненциальное изменение ускорение от одного значения к другому
- *PowerEase* Обеспечивает экспоненциальное изменение ускорения, при котором значение коэффициента ускорения пропорционально степени времени.
- *QuadraticEase* Подобно *CubicEase* за исключением того, что в данном случае значение пропорционально квадрату времени.
- *QuarticEase* Подобно *QuadraticEase* и *CubicEase*. В данном случае значение пропорционально времени в 3 степени.
- *QuinticEase* Подобно *QuadraticEase*, *CubicEase* и *QuarticEase*. В данном случае значение пропорционально времени в 5 степени
- *SineEase* Обеспечивает изменение значения по синусоиде.

Обратите внимание, что все это классы, т.е. они имеют соответствующие свойства, которые позволяют конфигурировать и настраивать эти режимы. Например, в используемом в предыдущем примере *BounceEase* есть свойства для описания количества колебаний и «прыгучести» анимации (например, изменение диапазона значений при изменении направления движения). При работе с функциями замедления всегда внимательно изучайте документацию по API, чтобы обеспечить необходимые эффекты.

Заключение

В данной главе вы узнали, как в Silverlight XAML описываются трансформации и анимации. Были представлены различные типы трансформации, используемые для вращения, изменения размеров или наклонения объекта, а также произвольные трансформации с использованием аффинной матрицы. Затем были рассмотрены анимации и их запуск с помощью триггера XAML. Показано, как анимация меняет значения свойств со временем, и представлены типы XAML, поддерживающие анимацию значений: *double*, *point* и *color*. Также мы научились использовать ключевые кадры для более тонкой

настройки анимации. Наконец, вы познакомились со средством графического создания анимации, Expression Blend, и увидели, как просто в нем создавать анимации визуально.

В Главе 6 будет рассмотрено использование элементов управления XAML для компоновки.

Глава 6

Элемент управления браузера Silverlight

В данной главе подробно рассматриваются вопросы разработки решений для Microsoft Silverlight, свойства и методы, предоставляемые программистам на JavaScript, использование тега <*object*> в HTML для создания экземпляра Silverlight и также использование этого объекта для выполнения Silverlight *вне* браузера.

При создании программ установки для приложений важно понимать объект Silverlight и применение его модели программирования в организации процесса установки для пользователя. В данной главе показан этот процесс и поведение объекта в различных браузерах и операционных системах, что позволит наилучшим образом спроектировать установку Silverlight для пользователей, не имеющих его.

Размещение Silverlight в браузере

Для использования и создания приложений Silverlight не требуется никакого специального программного обеспечения. Создавать сайты на Silverlight можно, используя любое ПО для разработки Веб-сайтов, начиная от Notepad и Eclipse, заканчивая Microsoft Expression Web или Expression Blend. Выбор исключительно за вами.

Данный раздел представляет базовое руководство по тому, как начать использовать Silverlight. До сих пор в этой книге всю сложную работу за вас выполнял шаблон Expression Blend или Microsoft Visual Studio, но теперь пришло время узнать, как самостоятельно создать и запустить простое приложение Silverlight, не используя никаких других инструментов, кроме Windows Explorer и Notepad.

Сначала создается HTML-файл. Элемент управления Silverlight будет размещаться на этой странице. Для этого используется тег <*object*>. Вот пример:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-
transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" >
<head>
   <title>App1</title>
</head>

<body>
  <div id='errorLocation' style="font-size: small;color: Gray;"></div>
  <div id="silverlightControlHost">
    <object type="application/x-silverlight-2"
      width="100%" height="100%" id="slPlugin">
        <param name="source" value="ClientBin/App1.xap"/>
        <param name="onerror" value="onSilverlightError" />
        <param name="background" value="white" />
        <param name="minRuntimeVersion" value="3.0.40624.0" />
        <param name="autoUpgrade" value="true" />
        <p>Silverlight isn't installed</p>
        </object>
        <iframe id='_sl_historyFrame' style='visibility:hidden;height:0;width:0;border:0px'></iframe>
   </div>
</body>
</html>
```

С помощью тега <*object*> создается экземпляр Silverlight. Это стандартный HTML-тег, следовательно, он поддерживает стандартные параметры. Мы не будем рассматривать здесь полный список параметров, остановимся лишь на используемых в предыдущем примере:

- Атрибут Type (Тип) определяет тип загружаемого объекта. В данном случае задано значение application/x-silverlight-2, которое указывает браузеру загружать подключаемый модуль Silverlight для доступа к этому содержимому.
- Атрибут Width (Ширина) определяет ширину элемента управления в процентном соотношении или в пикселах.
- Атрибут Height (Высота) определяет высоту элемента управления в процентном соотношении или в пикселах.
- Атрибут ID определяет имя элемента управления, используемое в коде JavaScript.

Объекты поддерживают разные наборы настраиваемых параметров, поэтому HTML-тег <*object*> поддерживает список элементов <*Param*>, позволяющие задавать нестандартные параметры. Эти элементы используются для определения пар имя/значение. В табл. 6-1 перечислены необязательные значения.

ТАБЛИЦА 6-1 **Параметры, поддерживаемые объектом Silverlight**

Имя параметра	Описание
Source	Определяет либо XAP, содержащий приложение для загрузки и выполнения, либо XAML-документ для формирования визуального представления.
Width	Устанавливает ширину элемента управления в пикселах или в процентном соотношении.
Height	Устанавливает высоту элемента управления в пикселах или в процентном соотношении.
Background	Определяет цвет фона элемента управления. Задание цветов подробно рассматривается в разделе «*SolidColorBrush*» Главы 4, «Основы Silverlight XAML». Можно использовать ARGB-значение, например #FFAA7700, или именованный цвет, например *Black*.
Framerate	Устанавливает максимально допустимую частоту кадров анимации. Значение по умолчанию равно 24.
isWindowless	Этот параметр может принимать значения *true* или *false*, значение по умолчанию – *false*. Если задано *true*, визуальное представление содержимого Silverlight формируется позади HTML-содержимого, так что HTML-содержимое может отображаться поверх его.
enableHtmlAccess	Определяет возможность доступа к содержимому DOM[1] браузера из кода, располагающегося в элементе управления Silverlight. Значение по умолчанию – *true*.
minRuntimeVersion	Определяет минимально необходимую поддерживаемую версию Silverlight.
onLoad	Определяет функцию, вызов которой будет

[1]Объектная модель документа (*прим. переводчика*).

	происходить при загрузке элемента управления.
onError	Определяет функцию, вызов которой будет происходить в случае возникновения ошибки.
onFullScreenChange	Это событие формируется при изменении свойства *FullScreen* (Во весь экран) элемента управления Silverlight.
onResize	Это событие формируется при изменении свойства *ActualWidth* или *ActualHeight* элемента управления Silverlight.

Замечательная особенность тега <*object*> в том, что он является наращиваемым, в том смысле, что если создать экземпляр исходного <*object*>не удалось, браузер на этом месте сформирует визуальное представление следующей части HTML, при условии что HTML располагается до закрывающего тега </*object*>. Поэтому, если Silverlight не установлен, на его место на экране без труда можно поставить простой баннер.

Рассмотрим пример:

```
<object data="data:application/x-silverlight,"
 type="application/x-silverlight-2"
 width="100%" height="100%">
 <param name="source" value="Page.xaml"/>
 <param name="onerror" value="onSilverlightError" />
 <param name="background" value="white" />

 <a href="..." style="text-decoration: none;">
  <img src="..." alt="Get Microsoft Silverlight"
     style="border-style: none"/>
 </a>
</object>
```

В данном случае в объект встроена гиперссылка (тег <*a*>), поэтому если Silverlight не установлен в системе, этот тег обеспечивает ссылку, по которой можно перейти на сайт загрузки Microsoft.

Реакция на события загрузки страницы

Возможны различные варианты обработки событий страницы для загрузки Silverlight.

Первый вариант – задать функцию JavaScript, которая будет вызываться в ответ на событие *Onload* (После загрузки) тега <*Body*> в HTML. Этот метод рекомендуется применять в случае выполнения проверки на возможность загрузки *Silverlight*, и не обязательно вашего приложения. Приложение Silverlight будет загружаться после выполнения вашей страницы, поэтому положительный ответ на загрузку страницы не означает возможности загрузки приложения Silverlight.

Второй вариант – задать обработчик событий JavaScript для управления событиями загрузки страницы с помощью параметра *onLoad*, представленного в табл. 6-1. Событие *onLoad* формируется после полной загрузки XAML-содержимого или XAP-файла элемента управления Silverlight. Обратите внимание, что если для какого-либо XAML-элемента UI будет определено аналогичное событие, оно будет сформировано до события *onLoad* элемента управления Silverlight. Кроме того, у элемента управления есть доступное только для чтения свойство *IsLoaded* (Загружен), которое устанавливается непосредственно перед формированием события *onLoad*.

При использовании обработчика события *onLoad* функция JavaScript должна принимать три параметра: первый – ссылка на элемент управления, второй – пользовательский контекст, и третий – ссылка на корневой элемент XAML. Например:

```
function handleLoad(control, userContext, rootElement)
{
  ...
}
```

Третий вариант ответа на загрузку страницы – использование выделенного кода в Silverlight, в котором определяются действия, выполняемые в ответ на загрузку либо приложения, либо любого XAML-элемента вашего приложения (например, страницы по умолчанию).

Для *Application* стандартный шаблон приложения Silverlight определяет *Application_Startup* как обработчик события в файле выделенного кода App.xaml.cs.

Для отдельных страниц можно задать обработчик событий в параметре *Loaded* в XAML, как в данном примере:

```
<UserControl x:Class="SLTestApp1.MainPage"
   xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
   xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
   Width="400" Height="300"
   Loaded="UserControl_Loaded">
   <Grid x:Name="LayoutRoot" Background="White">

   </Grid>
</UserControl>
```

В этом случае в выделенном коде страницы необходим метод с такой сигнатурой, или Silverlight сформирует ошибку:

```
private void UserControl_Loaded(object sender, RoutedEventArgs e)
{

}
```

Как альтернативный вариант, код можно выполнить в конструкторе страницы; если имя страницы MainPage.xaml, конструктор будет *MainPage()*.

Реакция на события, возникающие в случае ошибки на странице

Silverlight предоставляет несколько вариантов обработки ошибок в зависимости от типа ошибки. Ошибки возникают в случаях, когда синтаксический анализатор XAML сталкивается с проблемой, XAP-код формирует ошибку, загрузка не завершается соответствующим образом, возникает ошибка времени выполнения и если описанные в XAML-документе обработчики событий не имеют соответствующих JavaScript-функций или функций в выделенном коде.

При инициализации элемента управления с помощью обработчика событий *onError* задается функция JavaScript, которая будет вызываться при возникновении ошибки. Однако если такая функция не будет задана (или если она задана как *null*), то никакого уведомления об ошибке не будет происходить, и на месте элемента управления Silverlight будет отображаться пустая серая область. Но в проектах, созданных из Expression Blend, описывается стандартный обработчик ошибок.

Стандартный обработчик событий

Стандартный обработчик событий JavaScript обеспечит вывод на экран сообщения об ошибке с основными деталями возникшей ошибки Silverlight, включая код и тип ошибки, а также сообщение с указанием конкретной проблемы и имени вызванного метода.

Далее представлен пример неправильно сформированного XAML-документа, в котором сделана ошибка в имени закрывающего тега элемента *TextBlock*, он ошибочно назван *</TextBlok>*:

```
<Canvas xmlns="http://schemas.microsoft.com/client/2007"
      xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml">
  <TextBlock>Hello, World!</TextBlok>
</Canvas>
```

Если используется обработчик, созданный Expression Blend[1] по умолчанию, он обеспечит вывод на экран сообщения об ошибке Silverlight, как показан на рис. 6-1.

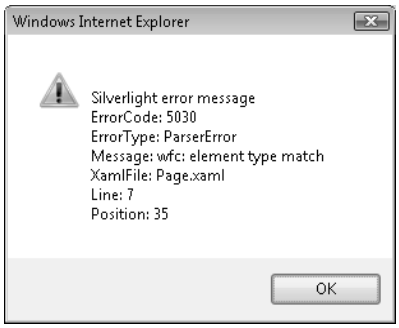

РИС. 6-1 Сообщение об ошибке.

Использование собственного обработчика ошибок

Можно использовать собственный обработчик ошибок. Для этого свойству *onError* элемента управления Silverlight необходимо задать специальную функцию обработчика ошибок. Она должна будет принимать два параметра: объект отправителя события и аргументы события, описывающие детали возникшей ошибки.

Существует три типа аргументов события. Первый тип – базовый объект *ErrorEventArgs* (Аргументы ошибки), содержащий тип сообщения об ошибке и код. Свойство *errorType* (Тип ошибки) определяет тип ошибки в виде строки, содержащей значения *RuntimeError* (Ошибка времени выполнения) или *ParserError* (Ошибка синтаксического анализатора). На основании этой информации используется один из двух производных типов ошибок.

При обработке в XAML ошибки синтаксического разбора используется объект *ParserErrorEventArgs* (Аргументы ошибки синтаксического анализатора). Оно содержит ряд свойств:

- Свойство *charposition* (позиция символа) содержит позицию символа, в котором возникла ошибка.
- Свойство *linenumber* (номер строки) содержит номер строки, в которой возникла ошибка.
- Свойство *xamlFile* (xaml-файл) определяет файл, в котором возникла ошибка.
- Свойство *xmlAttribute* (xml-атрибут) определяет XML-атрибут, в котором возникла ошибка.
- Свойство *xmlElement* (xml-элемент) определяет XML-элемент, в котором возникла ошибка.

Ошибки времени выполнения определяются в объекте *RuntimeErrorEventArgs* (Аргументы ошибки времени выполнения). Этот объект также включает ряд свойств:

- Свойство *charPosition* определяет позицию символа, в котором возникла ошибка.
- Свойство *lineNumber* определяет строку, в которой возникла ошибка.
- Свойство *methodName* (имя метода) определяет метод, связанный с этой ошибкой.

[1] Такое окно будет выводится только для приложения Silverlight 1. Для приложений Silverlight 2 и 3 при компиляции будет выполнена проверка исходного кода, и в панели Results (Результаты) будет выведено сообщение о допущенной ошибке (*прим. редактора*).

Рассмотрим, как ошибку можно перехватить собственным обработчиком ошибок. Прежде всего, создается тег *<object>*, определяющий обработчик ошибок:

```
<div id="silverlightControlHost">
  <object data="data:application/x-silverlight-2,"
      type="application/x-silverlight-2" width="100%" height="100%">
   <param name="source" value="doError.xaml"/>
   <param name="onerror" value="onSilverlightError" />
   <param name="background" value="white" />


   <param name="minRuntimeVersion" value="3.0.40624.0" />
   <param name="autoUpgrade" value="true" />
   </object>
   <iframe id='_sl_historyFrame' style='visibility:hidden;height:0;width:0;border:0px'></iframe>
</div>
```

В этом фрагменте кода описывается JavaScript-функция *onSilverlightError*, которая будет вызываться в ответ на любую ошибку.

Рассмотрим пример функции, которая обрабатывает и сообщает об ошибке:

```
   <script type="text/javascript">
      function onSilverlightError(sender, args) {

         var appSource = "";
         if (sender != null && sender != 0) {
            appSource = sender.getHost().Source;
         }
         var errorType = args.ErrorType;
         var iErrorCode = args.ErrorCode;

         var errMsg = "Unhandled Error in Silverlight Application " +  appSource + "\n" ;

         errMsg += "Code: "+ iErrorCode + "    \n";
         errMsg += "Category: " + errorType + "       \n";
         errMsg += "Message: " + args.ErrorMessage + "     \n";

         if (errorType == "ParserError")
         {
            errMsg += "File: " + args.xamlFile + "     \n";
            errMsg += "Line: " + args.lineNumber + "     \n";
            errMsg += "Position: " + args.charPosition + "     \n";
         }
         else if (errorType == "RuntimeError")
         {
            if (args.lineNumber != 0)
            {
               errMsg += "Line: " + args.lineNumber + "     \n";
               errMsg += "Position: " +  args.charPosition + "     \n";
            }
            errMsg += "MethodName: " + args.methodName + "     \n";
         }

         alert(errMsg);
      }
   </script>
```

Если при выполнении этого кода возникает ошибка, на экран будет выведено окно предупреждения с содержимым ошибки. На рис. 6-2 приведен пример собственного окна предупреждения.

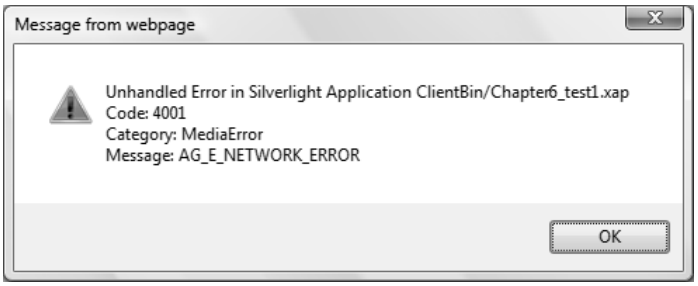

РИС. 6-2 Использование собственного обработчика ошибок.

Свойства элемента управления Silverlight

Элемент управления Silverlight имеет ряд свойств, некоторые из которых уже обсуждались в разделе «Размещение Silverlight в браузере». Свойства могут задаваться не только при инициализации элемента управления, но также с помощью сценария. Элемент управления разделяет свойства на три типа: непосредственные свойства, свойства содержимого и свойства настроек. *Непосредственные свойства* – это свойства самого элемента управления, которые доступны посредством синтаксиса *control.имясвойства*. Доступ к свойствам *содержимого* и *настроек* осуществляется посредством синтаксиса *control.content.имясвойства* и *control.settings.имясвойства*, соответственно[1].

Непосредственные свойства

Ниже перечислены поддерживаемые непосредственные свойства:

- ***initParams*** В этом свойстве хранятся параметры инициализации, передаваемые в элемент управления. Оно может быть задано только как часть инициализации элемента управления.
- ***isLoaded*** Свойство *isLoaded* принимает значение *true* после загрузки элемента управления; в противном случае, оно имеет значение *false*. Является доступным только для чтения.
- ***source*** Это XAML-содержимое, визуальное представление которого должно быть сформировано. Это может быть ссылка на файл, URI службы, формирующей XAML, или (в случае присутствия знака числа #) DIV, содержащий XAML-код в блоке сценария.

Свойства содержимого

Для организации доступа к свойствам содержимого может использоваться синтаксис *control.content.имясвойства*. Например, если необходимо обратиться к свойству *actualHeight*, используется синтаксис *control.content.actualHeight*. Доступны следующие свойства содержимого:

- ***actualHeight*** Возвращает высоту области отображения элемента управления Silverlightin в пикселах. Возвращаемое значение зависит от ряда критериев. Во-первых, от того, как высота элемента управления была задана изначально. Вспомним, что это может быть процентное соотношение или абсолютное значение в пикселах. В первом случае, свойство *actualHeight* возвращает текущую высоту элемента управления, но если пользователь изменит размеры браузера, ее значение изменится. Если высота задана абсолютным значением, будет возвращено это значение. При использовании элемента управления в полноэкранном режиме, это свойство будет возвращать разрешающую способность экрана по вертикали.
- ***actualWidth*** Возвращает ширину экрана. Возвращаемое значение зависит от ряда критериев и аналогично параметру *actualHeight*.

[1]При использовании функции CreateObject имена большинства свойств используются с маленькой буквы, но при программном обращении к ним во время выполнения первая буква в имени свойства должна быть большой. Это касается всех трех типов свойств (*прим. редактора*).

- ***fullScreen*** Переключает режим отображения элемента управления Silverlight между встроенным и полноэкранным режимами. По умолчанию имеет значение *false*, что обозначает встроенный режим. Когда этому свойству задано значение *true*, элемент управления Silverlight будет отображаться во весь экран.

Свойства настроек

Элемент управления Silverlight также имеет ряд свойств, которые определены как свойства настроек. Доступ к этим свойствам осуществляется посредством синтаксиса *control.settings.имясвойства*:

- ***background*** Свойство *background* задает цвет фона элемента управления Silverlight. Может принимать значения в нескольких форматах, включая именованный цвет (например, *Black*), 8-разрядные значения *Red/Green/Blue* (*RGB*) с или без альфа-канала и 16-разрядные *RGB*-значения с или без альфа-канала.
- ***enableFrameRateCounter (Счетчик частоты кадров включен)*** Если задано значение *true*, Silverlight будет отображать текущую частоту кадров (в кадрах в секунду) в строке состояния браузера. Значение по умолчанию – *false*.
- ***enableHtmlAccess (Html-доступ включен)*** Если задано значение *true*, объектная модель документов (Document Object Model, DOM) браузера становится доступной из Silverlight. Значение по умолчанию – *true*.
- ***enableRedrawRegions (Перерисовка областей включена)*** Если задано значение *true*, отображаются области объекта подключаемого модуля, перерисовываемые в каждом кадре. Это полезный инструмент оптимизации приложения. Значение по умолчанию – *false*.
- ***maxFrameRate (Максимальная частота кадров)*** Определяет максимальную частоту кадров генерируемого содержимого Silverlight. По умолчанию принимает значение 24 и имеет абсолютный максимум 64.
- ***version (версия)*** Сообщает об используемой в настоящий момент версии элемента управления Silverlight. Это строка, содержащая до четырех разделенных точками целых чисел: основной, дополнительный номера версий, номер сборки и номер редакции, – хотя обязательными являются только первые два значения (основной и дополнительный номера версии).
- ***windowless*** Это свойство определяет режим отображения элемента управления: безоконный или оконный. Если задано значение *true*, используется безоконный режим, т.е. содержимое Silverlight отображается на странице «за» HTML-содержимым.

Методы элемента управления Silverlight

Элемент управления Silverlight имеет ряд методов, которые могут использоваться для управления его поведением и функционированием. Аналогично группам свойств Silverlight, методы Silverlight тоже сгруппированы в «семейства» методов. В настоящее время поддерживаются один непосредственный метод и три метода содержимого. Что есть что, будет представлено в следующих разделах, включая примеры, демонстрирующие синтаксис этих методов и способ доступа к ним.

Метод *createFromXaml*

Метод *createFromXaml* (создать из Xaml) является методом содержимого Silverlight, позволяющий задавать XAML-содержимое, которое динамически добавляется в элемент управления Silverlight. Этот метод принимает два параметра. Первый – строка, содержащая XAML, который будет использоваться, и второй – параметр *namescope* (область имен). Если второй параметр имеет значение *true* (значением по умолчанию является *false*), в предоставленном XAML создаются уникальные ссылки *x:Name*, не конфликтующие с именами существующих XAML-элементов.

createFromXaml позволяет задавать ограничение для XAML: в добавляемом XAML должен быть всего один корневой узел. Поэтому, если необходимо добавить несколько элементов, убедитесь, что все они располагаются в одном узле, включающем элемент *Canvas*.

Кроме того, *createFromXaml* не добавляет XAML в элемент управления Silverlight до тех пор, пока он не будет добавлен в дочерний элемент одного из элементов *Canvas* элемента управления. Итак, при вызове *createFromXaml* вы получаете ссылку на узел, и эта ссылка затем используется для добавления этого узла в дерево визуального представления. Рассмотрим пример:

```
function handleLoad(control, userContext, sender)
{
  var xamlFragment = '<TextBlock Canvas.Top="60" Text="A new TextBlock" />';
  textBlock = control.content.createFromXaml(xamlFragment);
  sender.children.add(textBlock);
}
```

Здесь создается XAML-код для элемента управления *TextBlock* с текстом «A new TextBlock» (Новый TextBlock). Этот XAML-код затем используется для создания XAML-узла в содержимом элемента управления. Когда это будет выполнено, Silverlight возвратит ссылку на *TextBlock*. Эта ссылка добавляется в дерево визуального представления элемента управления Silverlight и используется для прорисовки *TextBlock*.

Метод *createFromXamlDownloader*

Метод *createFromXamlDownloader* (создать из загрузчика Xaml) является методом содержимого, используемым в паре с объектом *Downloader* (Загрузчик), который будет рассмотрен в данной главе позже. Этот метод принимает два параметра. Первый параметр – ссылка на объект *Downloader*, который загружает XAML-код или пакет, содержащий XAML-код. Второй параметр – имя пакета загружаемого содержимого, который должен использоваться. Если это .zip-файл, задается имя файла, содержащегося в .zip-архиве, в котором находится XAML-код, который предполагается использовать. Если загруженное содержимое не в пакете .zip, в качестве значения этого параметра должна быть задана пустая строка.

Метод *createObject*

Метод *createObject* (создать объект) является непосредственным методом, предназначенным для создания одноразового объекта для определенной функции. В Silverlight единственным таким поддерживаемым объектом является *Downloader*. Более подробно остановимся на этом в данной главе позже.

Метод *findName*

Метод содержимого *findName* (найти имя) позволяет выполнять поиск узла в XAML-коде по атрибуту *x:Name*. Если *findName* находит узел с заданным именем, он возвращает ссылку на него, в противном случае, возвращается *null*.

Объект *Downloader*

Элемент управления Silverlight предоставляет объект *Downloader*, который обеспечивает возможность загружать дополнительные элементы, используя функциональность асинхронной загрузки. Это позволяет загружать отдельные ресурсы или ресурсы в виде .zip-архива.

Свойства объекта *Downloader*

Объект *Downloader* поддерживает следующие свойства:

- ***downloadProgress (показатель процесса загрузки)*** Свойство *downloadProgress* обеспечивает нормализованное значение (между 0 и 1), которое представляет объем загруженного содержимого в процентном соотношении к его общему размеру, где значение 1 эквивалентно 100% загрузки.
- ***status (состояние)*** Свойство *status* принимает код состояния HTTP для текущего состояния процесса загрузки. Возвращает стандартный код состояния HTTP, например, «404» для «Not Found» (не найден) или «200» для «OK».
- ***statusText (текст состояния)*** Свойство *statusText* принимает текст состояния HTTP для текущего состояния процесса загрузки. Он соответствует коду состояния для свойства *status*. Для успешного запроса *status* будет равен «200», и *statusText* будет «OK». Для получения дополнительной информации по кодам состояния HTTP посмотрите стандартные коды HTTP, опубликованные W3C (*http://www.w3.org/Protocols/rfc2616/rfc2616-sec10.html*).
- ***uri*** Свойство *uri* содержит URI объекта, к которому в настоящий момент обращается загрузчик.

Методы объекта *Downloader*

Объект *Downloader* поддерживает следующие методы:

- ***abort (отменить)*** Отменяет текущую загрузку и возвращает все свойства в исходное состояние.
- ***getResponseText (получить текст ответа)*** Возвращает строковое представление загруженных данных. Принимает обязательный параметр, который используется для указания именованного блока в загружаемом пакете.
- ***open (открыть)*** Запускает сеанс загрузки. Принимает три параметра. Первый – команду действия. Набор методов HTTP задокументирован W3C; однако, Silverlight поддерживает только *GET* (Получить). Второй параметр – URI ресурса, который должен будет загружаться.
- ***send (отправить)*** Выполняет запрос загрузки, который был запущен командой *open*.

События объекта *Downloader*

Объект *Downloader* поддерживает следующие события:

- ***completed (завершен)*** Это событие формируется по завершении загрузки. Оно принимает два параметра. Первый – объект, сформировавший это событие (в данном случае, сам элемент управления downloader), и второй – набор аргументов события (*eventArgs*).
- ***downloadProgressChanged (показатель процесса загрузки изменился)*** Это событие формируется в ходе загрузки. Оно формируется при каждом изменении показателя процесса загрузки (значение которого меняется от 0 до 1) на 0,05 (5%) или более, а также когда он достигает значения 1,0 (100%). По достижении 1,0 также формируется событие *completed*.

Использование объекта *Downloader*

Объект *Downloader* позволяет выполнять доступ к ресурсам сети из JavaScript. Пожалуйста, обратите внимание, что этот объект будет работать, только если приложение Silverlight выполняется с Веб-сервера, и будет формировать ошибку в случае загрузки страницы просто из файловой системы.

Объект *Downloader* создается методом *createObject*, предоставляемым элементом управления Silverlight. Рассмотрим пример:

```
<script type="text/javascript">
   function handleLoad(control, userContext, sender)
   {
     var downloader = control.createObject("downloader");
   }
</script>
```

Следующий шаг – запустить сеанс загрузки. Для этого с помощью метода *open* объекта *Downloader* объявляется URI файла и вызывается метод *send* для начала загрузки. Далее представлен пример, в котором выполняется загрузка файла фильма *movie.wmv*:

```
function handleLoad(control, userContext, sender)
{
  var downloader = control.createObject("downloader");
  downloader.open("GET","movie.wmv");
  downloader.send();
}
```

Для отслеживания процесса загрузки и момента ее завершения понадобится создать соответствующие обработчики событий. Далее представлена та же функция, но в нее были внесены необходимые дополнения:

```
function handleLoad(control, userContext, sender)
{
  var downloader = control.createObject("downloader");
  downloader.addEventListener("downloadProgressChanged","handleDLProgress");
  downloader.addEventListener("completed","handleDLComplete");
  downloader.open("GET","movie.wmv");
  downloader.send();
}
```

Теперь можно применить эти обработчики событий. В данном примере событие *downloadProgressChanged* связано с функцией JavaScript *handleDLProgress*, и событие *completed* – с функцией JavaScript *handleDLComplete*. Эти функции представлены в данном фрагменте кода:

```
function handleDLProgress(sender, args)
{
  var ctrl = sender.getHost();
  var t1 = ctrl.content.findName("txt1");
  var v = sender.downloadProgress * 100;
  t1.Text = v + "%";
}

function handleDLComplete(sender, args)
{
   alert("Download complete");
}
```

> **Примечание** Объект *Downloader* может использоваться только для приложений, размещенных на Веб-сервере. В случае попытки его использования со страницы, загружаемой из файловой системы, будет сформирована ошибка.

Также обратите внимание, что объект *Downloader* позволяет загружать ресурсы, использующие JavaScript. Для приложений Silverlight, которые работают с Microsoft .NET Framework, существует огромное количество вариантов доступа к сетевым ресурсам. Более подробно эти вопросы рассматриваются в Главе 12, «Создание взаимодействующих с сервером приложений в Silverlight».

Использование объектной модели для проектирования процесса установки

Объектная модель Silverlight предоставляет события, которые позволяют создавать очень удобные и понятные для пользователей процессы установки Silverlight.

Когда возникает необходимость в использовании Веб-сайта, которому для формирования представления содержимого требуется подключаемый модуль, обычно перед пользователем возникает ряд вопросов. Стоит ли это содержимое того времени и усилий, которые будут затрачены

на установку этого подключаемого модуля? Понимаю ли я, что устанавливаю? *Хочу* ли я это установить?

Проведя исследования популярных Веб-сайтов, таких как NBC Olympics и Democratic National Conference, а также огромного числа собственных сайтов Microsoft, Microsoft был выявлен ряд основных факторов, которые облегчают это решение для пользователя, упрощают процесс установки и обеспечивают с наибольшей вероятностью то, что пользователь предпочтет загрузить подключаемый модуль для взаимодействия с вашим содержимым.

В данном разделе описаны возможные сценарии установки Silverlight и приведен код для решения любых проблем, которые могут возникать в этих сценариях. На самом деле, проблем больше, чем кажется. Пользователь может не иметь Silverlight; у пользователя может быть старая версия; возможны состояния, когда потребуется перезапустить браузер и когда достаточно будет лишь обновления страницы.

Программирование процесса установки

В проектах Silverlight, создаваемых с помощью Silverlight Tools для Visual Studio, пользователям, не имеющим среды выполнения Silverlight, предлагается установка по умолчанию (посредством так называемой эмблемы Silverlight). Это очень простой универсальный процесс.

Важно отметить, что благодаря своей универсальности, этот процесс не привязан ни к одному конкретному приложению. Поэтому присутствие эмблемы установки может привести пользователей в замешательство.

Рассмотрим сценарий посещения пользователем игрового сайта. Пользователь выбирает игру, но вместо нее в верхнем левом углу экрана появляется эмблема Silverlight. Что это? Баннерная реклама? Я нажал не ту кнопку? В такой ситуации, когда пользователь пребывает в замешательстве, велика вероятность того, что он не щелкнет эмблему, не загрузит Silverlight и не будет играть в эту игру.

Но даже если пользователь щелкает эмблему, это не приводит к установке приложения, всего лишь запускается загрузка исполняемого приложения, взаимодействие с которым обеспечит пользователю установку подключаемого модуля. И после установки нет никакой гарантии, что содержимое будет работать правильно. Иногда для этого достаточно обновления страницы, иногда необходимо перезапустить браузер. Чтобы обеспечить пользователю наилучшее взаимодействие, необходимо эффективно и последовательно обработать все возможные сценарии.

Для этого требуется определиться с двумя основными моментами. Первое – это *технология* и код, которые могут обеспечить наилучшее взаимодействие с пользователем при установке. Второе – *приглашение к установке*, соответствующий язык и формулировки, абсолютно понятные пользователям, которые сделают процесс установки простым и прозрачным.

Прежде чем заняться кодом, не лишним будет разобраться со всеми возможными сценариями установки.

Спроектировав приложение установки, Веб-разработчик должен создать несколько вариантов, чтобы учесть также особые состояния браузера пользователя при посещении приложения с поддержкой Silverlight. Эти варианты отличаются только текстовой частью, но обеспечат доверие пользователя приложению и платформе.

Существует шесть возможных состояний пользователя, работающего с приложением установки Silverlight через Веб-сайт. Варианты предлагаемых пользователю сообщений должны быть семантически эквивалентны рекомендациям, приведенным в табл. 6-2.

ТАБЛИЦА 6-2 Состояния установки Silverlight

Состояние	Тип сообщения
Silverlight установлен и загружен.	Содержимое отображается без всяких предварительных сообщений.
Необходимая версия Silverlight установлена, но не загружена.	Поблагодарить пользователя, попросить перезапустить браузер и вернуться на эту страницу.
Установлена старая версия Silverlight.	Сообщить пользователю о необходимости установки последней версии Silverlight и направить его по адресу, где он может получить эту версию.
Выполняется обновление.	Сообщить пользователю о выполнении обновления и о необходимости перезапустить браузер по его завершении.
Silverlight не установлен.	Сообщить пользователю о том, что для получения содержимого ему необходимо установить подключаемый модуль Silverlight. Внимание должно акцентироваться на содержимом, не на Silverlight.
Выполняется установка.	Поблагодарить пользователя за установку Silverlight и сообщить о возможной необходимости обновления браузера по завершении установки.
Неподдерживаемая конфигурация.	Предоставить пользователю сообщение о невозможности установки Silverlight в его среде. Это может быть сообщение, обсуждающее вопросы совместимости, или альтернативная HTML-страница.

Визуальное представление этих состояний предлагается на рис. 6-3.

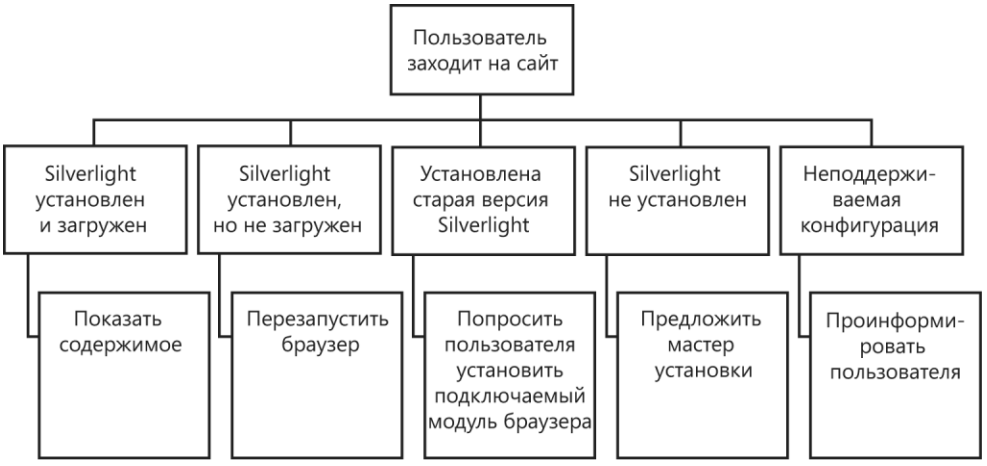

РИС. 6-3 Различные состояния установки.

Эти состояния и ожидания пользователя обрабатываются в коде с применением следующих этапов

Задача 1. Интегрирование Silverlight.js и использование тега *<object>*

Шаг 1. Интегрировать Silverlight.js

Silverlight.js должен быть включен в элемент страницы *<head>* с помощью тега *<script>*. Этот JavaScript-файл определяет необходимые API и содержит код, обеспечивающий проверку наличия установленного Silverlight.

```
<head>
<script src="Silverlight.js" type="text/javascript" ></script>
</head>
```

Шаг 2. Использовать тег *<object>* для Silverlight

Как было показано ранее, встраивание Silverlight в Web-страницу осуществляется с помощью HTML-тега *<object>*. Этот тег должен включать атрибуты *data* (дата) и *type* (тип). Обратите внимание, что по умолчанию проекты Silverlight в качестве MIME-типа для этих атрибутов используют «*application/x-silverlight-2*».

Чтобы определить состояние установки для пользователей Silverlight 1.0, следует использовать MIME-тип Silverlight 1.0 как для атрибута *data*, так и для *type*. Тем самым вы вынуждаете пользователя обновить элемент управления, избавляя при этом от беспокойства автора Веб-сайта.

Итак, эти атрибуты должны быть описаны следующим образом:

```
type="application/x-silverlight"
```

и

```
data="data:application/x-silverlight,"
```

Шаг 3. Использовать параметр *minRuntimeVersion*

Параметр *minRuntimeVersion* (минимальная версия) – часть тега *<object>*, позволяющая гарантировать, что у пользователя установлена соответствующая версия Silverlight. Если установлена не та версия, XAP загружаться не будет, а пользователю будет предложено обновить элемент управления. Кроме того, это состояние обеспечит формирование события *OnError*, *args.ErrorCode* которого будет иметь значение 8001 «Upgrade Required» (Требуется обновить) или 8002 «Restart Required» (Требуется перезапустить).

Обратите внимание, что поскольку Silverlight 1.0 не поддерживал параметр *minRuntimeVersion*, проверки состояния ошибки не достаточно для определения необходимости выполнения некоторого действия на стороне пользователя. Выявить такую необходимость можно, только следуя шагам, приведенным в данной главе, и используя Silverlight.js.

Для RTM-версии Silverlight 2 параметр *minRuntimeVersion* должен иметь значение «2.0.31005». RTM-версия Silverlight 3 на момента написания данной книги недоступна, но эта информация появится после публикации книги. Все подробности можно найти в блоге автора этой книги по адресу *http://blogs.msdn.com/webnext*.

Шаг 4. Использовать параметр *autoUpgrade*

Если параметру *autoUpgrade* (автоматическое обновление) задано значение *true*, выполняется сценарий обновления Silverlight по умолчанию. Если вы хотите самостоятельно управлять этим процессом с помощью собственного мастера обновлений, задайте этому параметру значение *false*.

Тестировать свой мастер обновлений можно, создав объект Silverlight на странице с использованием следующих значений свойств *data*, *type*, *minRuntime* и *autoUpgrade*:

```
<object data="data:application/x-silverlight,"
type="application/x-silverlight">
<param name="minRuntimeVersion" value="9.9.99999" />
<param name="autoUpgrade" value="true" />
</object>
```

Шаг 5. Включить обработку ошибок

В теге <*object*> должен использоваться параметр *onError*, и в функции-обработчике необходимо вызывать *Silverlight.IsVersionAvailableOnerror(sender,args)*.

Если эта функция возвращает *true*, значит, необходимая версия Silverlight уже установлена *и* загружена, т.е. должен выполняться обычный код обработки ошибки.

Если в теге <*object*> параметр описан так:

```
<param name="onError" value="onSilverlightError" />
```

то обработчик должен выглядеть следующим образом:

```
function onSilverlightError(sender, args)
{
  if (Silverlight.IsVersionAvailableOnError(sender, args)) {
     // выполняем код обработки ошибки
  }
}
```

Шаг 6. Обработать событие *onLoad*

Параметр *onLoad* тега <*object*> должен определять функцию-обработчик, и эта функция должна вызывать *Silverlight.IsVersionAvailableOnLoad(sender)*. Вызов этой функции позволяет выявить предыдущие версии Silverlight (например, бета-версии) и запустить обновление или перезапуск для обработки этого состояния.

Итак, для такого параметра тега <*object*>:

```
<param name="onLoad" value="onSilverlightLoad" />
```

обработчик должен выглядеть следующим образом:

```
function onSilverlightLoad(sender)
{
  Silverlight.IsVersionAvailableOnLoad(sender);
}
```

Шаг 7. Обработать ситуацию несовместимости конфигураций

Хотя элемент управления Silverlight доступен для установки большинству пользователей Интернета, есть платформы, которые не поддерживают его в настоящий момент. Когда пользователи таких платформ щелкают URL установки Silverlight, они перенаправляются на Веб-страницу по адресу *http://www.microsoft.com/silverlight*, где им сообщается о том, что их платформа не поддерживается. Такой сценарий вполне удовлетворителен для многих страниц, но некоторые Веб-сайты хотят иметь возможность выявлять таких пользователей без перенаправления на *http://www.microsoft.com*. Это можно реализовать несколькими способами:

- Идентифицировать пользователя неподдерживаемого клиента на основании HTTP-запроса
- Идентифицировать пользователя неподдерживаемого клиента на основании *navigator.userAgent*

Стандартная реализация выявления поддерживаемого клиента предлагается в Silverlight.supportedUserAgent.js (*http://code.msdn.microsoft.com/SLsupportedUA*).

Рассмотрим один из возможных сценариев использования по выявлению пользователя поддерживаемого клиента.

Сначала добавим ссылку на файл сценария на страницу:

```
<script type="text/javascript" src="Silverlight.supportedUserAgent.js"></script>
```

Затем создадим HTML-приглашение для неподдерживаемого сценария и загрузим его в JavaScript *var*:

```
var PromptNotSupported = "<div><p>К сожалению, данный браузер не поддерживаетSilverlight.</p></div>";
```

Наконец, создадим функцию JavaScript, которая будет использовать API *supportedUserAgent* для определения того, поддерживается ли текущий браузер. Обратите внимание, чтобы это сработало, объект Silverlight должен находиться в <*div*> под именем *silverlightControlHost*.

```
    <script type="text/javascript">
function CheckSupported() {
  var tst = Silverlight.supportedUserAgent();
  if (tst) {
    // Не делаем ничего
  }
  else{
    document.getElementById("silverlightControlHost").innerHTML =
        PromptNotSupported;
  }
}
</script>
```

Чтобы использовать это, данный сценарий необходимо вызвать при загрузке страницы.

```
    <body onload="CheckSupported()">
```

Важно отметить, что Веб-браузеры и Silverlight развиваются, и по мере их изменения будет меняться и код выявления поддерживаемого агента пользователя. Следовательно, только частые проверки обновлений Веб-страницы Silverlight.supportedUserAgent обеспечат Веб-разработчикам использование самой последней версии Silverlight.

Задача 2. Обеспечение вывода приглашений UI в соответствующие моменты

На рис. 6-4 представлены возможные состояния установки. Исходя из них, видим, что необходимо пять основных приглашений.

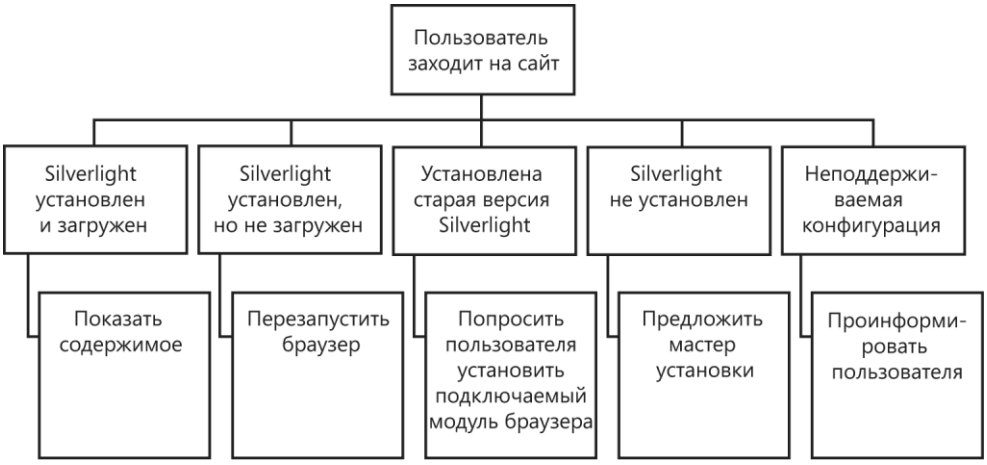

РИС. 6-4 Различные состояния установки.

Первый слева вариант – простейшее состояние: необходимая версия Silverlight установлена и запущена, поэтому просто отображаем содержимое.

Следующее состояние – выявлена необходимая версия Silverlight, но она не загружена. В этом случае пользователя просят перезапустить браузер.

Далее идет состояние, когда у пользователя *есть* Silverlight, но старая его версия. В этом случае пользователю предлагается кнопка Click to Upgrade (Обновить) и по завершении установки его просят перезапустить браузер.

Следующее состояние – Silverlight не установлен. В этом случае пользователю сначала предлагается кнопка Click to Install (Установить) и по завершении установки его просят обновить страницу в браузере.

Последнее состояние – выявлена несовместимая конфигурация.

Обратите внимание, что переходные состояния, когда требуется установить или обновить Silverlight, автоматически не выявляются. Соответствующие им приглашения необходимо формировать в ответ на нажатие пользователем кнопок Click to Install или Click to Upgrade. Как это делается, будет показано позже.

Итак, обобщаем всю изложенную информацию и получаем шесть разных вариантов подсказок пользователю, соответственно шести возможным состояниям:

- Silverlight не установлен; нажмите, чтобы установить.
- Идет установка Silverlight; обновите страницу в браузере по завершении.
- Установлена старая версия Silverlight; нажмите, чтобы обновить.
- Выполняется обновление Silverlight; перезапустите браузер по завершении.
- Перезапустите браузер.
- Ваш браузер несовместим с данным приложением.

Далее поэтапно рассмотрим создание мастера установки Silverlight.

Шаг 1. Определяем HTML, предлагающий пользователю установить Silverlight

В описание тега <*object*> можно включить HTML, отображаемый в случае невозможности создания объекта. Он будет выполняться, если Silverlight не установлен. На этом этапе все просто: после всех тегов <*param*> добавляем <*div*>, который формирует визуальное представление HTML.

Заметьте, что где-то в этом HTML необходимо фиксировать событие щелчка и использовать его для запуска JavaScript-функции. Сценарий необходим для формирования представления сообщения об установке.

Рассмотрим пример законченного тега <*object*> с таким HTML:

```
<object data="data:application/x-silverlight," type="application/x-silverlight-2"
        width="100%" height="100%">
<param name="source" value="bin/debug/Memory.xap"/>
<param name="background" value="white" />
<param name="minRuntimeVersion" value="3.0.40624.0" />
<param name="autoUpgrade" value="false" />
<param name="onerror" value="onSilverlightError" />
<param name="onload" value="onSilverlightLoad" />
<div id="SLInstallFallback" class="silverlightInstall" >
  <img src="images/install.PNG" onclick='InstallClicked();'
     class="silverlightInstallImage"
     alt='Click to Install' style='cursor:pointer;'/>
</div>
</object>
```

Этот код определяет отображение изображения (images/install.PNG) во время установки Silverlight. Приведенный HTML-код можно заменить собственным, соответственно ситуации. Однако не забывайте об обработчике события *onclick*. Обработчики рассматриваются в шаге 3.

Шаг 2. Определяем UI для приглашений других состояний

Приглашения для остальных пяти состояний могут быть определены как переменные JavaScript.

В данном примере используются такие переменные для состояний:

- ***var PromptFinishInstall*** Используется для состояния «Идет установка Silverlight; обновите по завершении».

- ***var PromptUpgrade*** Используется для состояния «Установлена старая версия Silverlight; нажмите, чтобы обновить».
- ***var PromptFinishUpgrade*** Используется для состояния «Выполняется обновление Silverlight; перезапустите по завершении».
- ***var PromptRestart*** Используется для состояния «Перезапустите браузер».
- ***var PromptIncompatible*** Используется для состояния «Ваш браузер несовместим с данным приложением».

Для этого просто используются HTML-объекты * *, которые замещают *innerHTML* части страницы, когда необходимо вывести подсказку.

Заметьте, что *PromptUpgrade* (аналогично приглашению в Шаге 1) должен иметь вызов JavaScript-функции, которая вызывается по щелчку приглашения. Эта JavaScript-функция обеспечит изменение приглашения на «Выполняется обновление Silverlight; перезапустите по завершении». Это будет показано в Шаге 3.

Рассмотрим пример вызова JavaScript-функции. Обратите внимание на описание обработчика события *onclick*:

```
  var PromptUpgrade =
"<div id='SLInstallFallback' class='silverlightInstall' >" +
"   <img src='images/upgrade.PNG' onclick='UpgradeClicked();'
        class='silverlightInstallImage'" +
"       alt='Click to Upgrade' style='cursor:pointer;'/>" +
"</div>";
```

Шаг 3. Задаем JavaScript-функции для обработки действий пользователя по установке и обновлению

В Шаге 1 описывалось приглашение щелкнуть определенную область страницы для установки Silverlight. В ответ на это действие загружается программа установки и обновляется страница, информируя пользователя о состоянии установки.

Это обеспечит следующий код на JavaScript:

```
   function InstallClicked()
{
 window.location.href = "http://go.microsoft.com/fwlink/?LinkID=149156&v=3.0.40624.0";
 document.getElementById("silverlightControlHost").innerHTML =
   PromptFinishInstall;
}
```

Обратите внимание на изменение значения *innerHTML* тега *<object>* на значение *PromptFinishInstall*, как описывалось в Шаге 2.

В Шаге 2 при описании приглашения состояния обновления был задан обработчик события *onClick UpgradeClicked*. Он аналогичен обработчику события *InstallClicked* в том смысле, что обеспечивает загрузку программного обеспечения и смену приглашения, но, в этот раз, на сообщение о завершении обновления.

```
   function UpgradeClicked()
{
 window.location.href = "http://go.microsoft.com/fwlink/?LinkID=149156&v=3.0.40624.0";
 document.getElementById("silverlightControlHost").innerHTML =
   PromptFinishUpgrade;
}
```

Задача 3. Обработка перехвата обратных вызовов из Silverlight.js

Silverlight.js имеет несколько вспомогательных функций, которые перехватывают различные состояния, что позволяет обновлять UI соответственно.

Это функции:

- *Silverlight.onRequiredVersionAvailable* Необходимая версия Silverlight установлена и доступна. С помощью этой функции можно сообщать пользователям о том, что они готовы к просмотру, или ее можно просто игнорировать и формировать визуальное представление содержимого.
- *Silverlight.onRestartRequired* Пользователь должен перезапустить браузер. В этом случае HTML области приглашения заменяется содержимым переменной *PromptRestart*.
- *Silverlight.onUpgradeRequired* Пользователь должен обновить старую версию Silverlight. В этом случае HTML области приглашения заменяется содержимым переменной *PromptUpgrade*.
- *Silverlight.onInstallRequired* Silverlight не установлен, но необходим. Вы не должны предпринимать здесь какие-либо действия, потому что это состояние перехватывается в теге <*object*>.

Рассмотрим код:

```
function onSilverlightLoad(sender)
{
  Silverlight.IsVersionAvailableOnLoad(sender);
}

Silverlight.onRequiredVersionAvailable = function()
{
};

Silverlight.onRestartRequired = function()
{
  document.getElementById("silverlightControlHost").innerHTML =
    PromptRestart;
};

Silverlight.onUpgradeRequired = function()
{
  document.getElementById("silverlightControlHost").innerHTML =
    PromptUpgrade;
};

Silverlight.onInstallRequired = function()
{
};
```

Задача 4. Обработка известных проблем

Известно, что с проблемами сталкиваются пользователи Firefox при обновлении версии Silverlight 1.0 или бета-версии Silverlight 2 до выпущенной версии Silverlight 2. Эта проблема решена в последующих версиях Silverlight, но для Silverlight 2 ее необходимо обрабатывать.

Пользователи Firefox, если не выполнят перезапуск своего браузера после обновления до Silverlight 2, будут снова видеть приглашение установить Silverlight 2, определенное в теге <*object*>.

Исправить это можно, применяя следующий сценарий *после* описания <*object*>.

```
<script type="text/javascript">
  try
  {
    if (navigator.plugins["Silverlight Plug-In"].description)
    {
      document.getElementById("SLInstallFallback").innerHTML = PromptRestart;
    }
  }
  catch (e)
  {
```

```
 }
</script>
```

Другая проблема в том, что для пользователей Mac Firefox ошибочно определяется состояние необходимости обновления, когда они на самом деле находятся в состоянии необходимости перезапуска. На данный момент для этой проблемы нет решения, но также и вероятность ее возникновения ничтожно мала.

Окончательный вариант мастера установки

Microsoft Visual Studio создает HTML-страницу по умолчанию для размещения Silverlight-содержимого. Следуя изложенным выше шагам, вы без труда измените стандартное поведение при установке на собственное, в котором будут использоваться продемонстрированные состояния установки.

Листинг 6-1 представляет окончательный код страницы, новый раздел в нем выделен жирным шрифтом.

ЛИСТИНГ 6-1 Окончательный вариант мастера установки

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-
transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" >
<!-- saved from url=(0014)about:internet -->
<head>
   <title>SilverlightApplication1</title>

   <style type="text/css">
   html, body {
            height: 100%;
            overflow: auto;
   }
   body {
            padding: 0;
            margin: 0;
   }
   #silverlightControlHost {
            height: 100%;
   }
   </style>
   <script type="text/javascript" src="Silverlight.js"></script>
   <script type="text/javascript" src="Silverlight.supportedUserAgent.js"></script>
   <script type="text/javascript">
      var PromptFinishInstall = "<div><p>Идет установка Silverlight, обновите браузер по
завершении.</p></div>";
      var PromptUpgrade = "<div><p onclick='UpgradeClicked()'>Данному приложению необходима Ваша
помощь для обновления Silverlight. У Вас установлена более старая версия. Нажмите, чтобы обновить.
</p></div>";
      var PromptFinishUpgrade = "<div><p>Выполняется обновление Silverlight; перезапустите браузер по
завершении .</p></div>";
      var PromptRestart = "<div><p> Пожалуйста, перезапустите браузер.</p></div>";
      var PromptNotSupported = "<div><p>К сожалению, Ваш браузер не поддерживает
Silverlight</p></div>";
      function onSilverlightError(sender, args) {

         if (Silverlight.IsVersionAvailableOnerror(sender, args)) {
            var appSource = "";
            if (sender != null && sender != 0) {
               appSource = sender.getHost().Source;
            }
            var errorType = args.ErrorType;
            var iErrorCode = args.ErrorCode;

            var errMsg = "Необработанная ошибка в приложении Silverlight 2 " + appSource + "\n";

            errMsg += "Code: " + iErrorCode + "    \n";
```

```
            errMsg += "Category: " + errorType + "       \n";
            errMsg += "Message: " + args.ErrorMessage + "     \n";

            if (errorType == "ParserError") {
               errMsg += "File: " + args.xamlFile + "     \n";
               errMsg += "Line: " + args.lineNumber + "     \n";
               errMsg += "Position: " + args.charPosition + "     \n";
            }
            else if (errorType == "RuntimeError") {
               if (args.lineNumber != 0) {
                  errMsg += "Строка: " + args.lineNumber + "     \n";
                  errMsg += "Позиция: " + args.charPosition + "     \n";
               }
               errMsg += "Имя метода: " + args.methodName + "     \n";
            }

            throw new Error(errMsg);
         }
      }

      function onSilverlightLoad(sender) {
         Silverlight.IsVersionAvailableOnLoad(sender);

      }

      Silverlight.onRequiredVersionAvailable = function() {
   };

      Silverlight.onRestartRequired = function() {
         document.getElementById("silverlightControlHost").innerHTML = PromptRestart;
      };

      Silverlight.onUpgradeRequired = function() {
         document.getElementById("silverlightControlHost").innerHTML = PromptUpgrade;
      };
      Silverlight.onInstallRequired = function() {
      };
      function UpgradeClicked() {
         window.location = "http://go2.microsoft.com/fwlink/?linkid=124807";
         document.getElementById("silverlightControlHost").innerHTML = PromptFinishUpgrade;
      }
      function InstallClicked() {
         window.location = "http://go2.microsoft.com/fwlink/?linkid=124807";
         document.getElementById("silverlightControlHost").innerHTML = PromptFinishInstall;
      }
         function CheckSupported() {
         var tst = Silverlight.supportedUserAgent();
         if (tst) {
            // Не делаем ничего
            }
         else{
            document.getElementById("silverlightControlHost").innerHTML = PromptNotSupported;
         }
      }

   </script>
</head>

<body onload="CheckSupported()">
   <!—Здесь будут отображаться ошибки времени выполнения Silverlight.
         Будет включать отладочную информацию и должно быть удалено или скрыто по завершении отладки -->
         <div id='errorLocation' style="font-size: small;color: Gray;"></div>

   <div id="silverlightControlHost" style="height:100%;">
                  <object data="data:application/x-silverlight," type="application/x-silverlight-2" width="100%"
height="100%">
                              <param name="source" value="ClientBin/SilverlightApplication1.xap"/>

                              <param name="background" value="white" />
                              <param name="minRuntimeVersion" value="3.0.40624.0" />
```

```
                        <param name="autoUpgrade" value="false" />
                        <param name="onerror" value="onSilverlightError" />
                        <param name="onload" value="onSilverlightLoad" />
                        <div id="SLInstallFallback"><div><p onclick='UpgradeClicked()'>Данному
приложению для использования Silverlight необходима ваша помощь по его установке. Нажмите, чтобы
установить. </p></div></div>
                    </object>
                    <iframe style='visibility:hidden;height:0;width:0;border:0px'></iframe>
  </div>
  <div id="silverlightExperienceHost" style="visibility:hidden;">
  </div>
  <script type="text/javascript">
    try {
      if (navigator.plugins["Silverlight Plug-In"].description) {
        document.getElementById("SLInstallFallback").innerHTML = PromptRestart;
      }
    }
    catch (e) {
    }
  </script>
</body>
</html>
```

В данном сценарии создан код, но не приглашение к установке. Рекомендуется тщательно проработать приглашения для собственного приложения, уделяя при этом основное внимание содержимому приложения, а не Silverlight. Так, например, на сайте рекламы мини-приложения, должна быть большая красивая крупная надпись, описывающая достоинства этого приложения, с коротким замечанием, обозначенным меньшим шрифтом, о том, что для доступа к мини-приложению необходимо установить подключаемый модуль, что является быстрой, простой и безопасной процедурой. Это можно видеть на существующих сайтах. Хороший пример приглашения к установке вы найдете по адресу *http://movies.msn.com/pretty-in-ink/* (рис. 6-5).

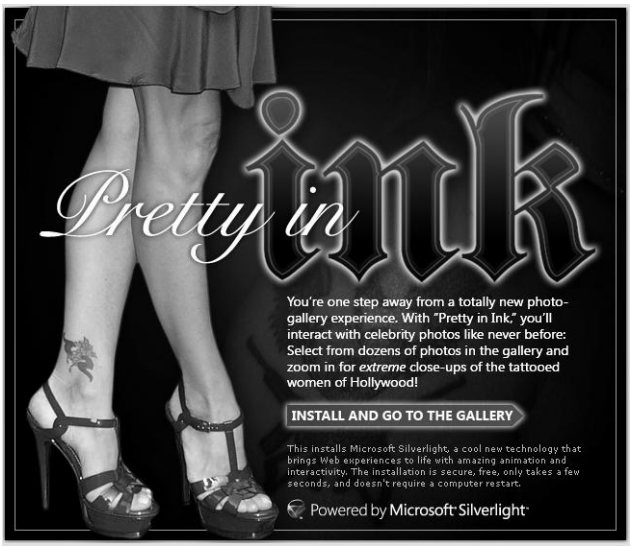

РИС. 6-5 Правильно спроектированное приглашение к установке.

Правильный дизайн приглашения к установке вместе с соответствующим набором состояний установки обеспечат меньшее количество отказов пользователей от установки и большее количество просмотров нового насыщенного Silverlight-содержимого.

Выполнение приложений Silverlight вне браузера

В Silverlight 3 можно разрабатывать приложения, выполняющиеся вне браузера. Это позволяет создавать приложения с собственным окном, которые могут быть добавлены в меню Пуск или на

Рабочий стол. Сделать это очень просто, как будет показано через мгновение: необходимо лишь ввести некоторые настройки в манифест приложения, определяющие возможность его выполнения в автономном режиме.

Создадим новое Silverlight-приложение с названием SlOOB. Добавим на страницу по умолчанию что-то простенькое, например, текстовый блок «Hello World», и выполним приложение. Оно будет выполнено в браузере, как обычно. Однако по щелчку содержимого Silverlight правой кнопкой мыши появится меню, в котором можно увидеть новый элемент, Install … Onto This Computer (Установить … на этот компьютер), неактивный в данном случае. В этой опции меню используется имя приложения (рис. 6-6), как настроить это рассмотрим чуть ниже.

РИС. 6-6 Возможность выполнения вне браузера выключена.

Чтобы включить эту функциональность, необходимо внести некоторые изменения в манифест приложения. Файл AppManifest.xml находится в папке Properties.

```
<Deployment xmlns="http://schemas.microsoft.com/client/2007/deployment"
      xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml">
   <Deployment.Parts>
   </Deployment.Parts>
</Deployment>
```

Первым делом, добавим параметры *EntryPointAssembly* (Сборка точки входа) и *EntryPointType* (Тип точки входа). Для приложения SLOOB это должны быть *SLOOB* и *SLOOB.App*, соответственно.

Теперь манифест должен выглядеть следующим образом:

```
<Deployment xmlns="http://schemas.microsoft.com/client/2007/deployment"
      xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
         EntryPointAssembly="SLOOB"
         EntryPointType="SLOOB.App">

   <Deployment.Parts>
   </Deployment.Parts>
</Deployment>
```

Далее необходимо добавить в манифест раздел *OutOfBrowserSettings*(Настройки поведения вне браузера). В нем будут располагаться краткое имя, заголовок и аннотация приложения.

- *Title* (Заголовок) отображается в строке заголовка отдельного окна, в котором выполняется приложение.
- *Short name* (Краткое имя) отображается на ярлыках рабочего стола и/или меню Пуск.
- *Blurb* (Аннотация) используется в разделе Comments (Комментарии) для файла. Его можно увидеть при просмотре свойств приложения в Windows Explorer.

Все эти элементы представлены в манифесте:

```
<Deployment xmlns="http://schemas.microsoft.com/client/2007/deployment"
      xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
         EntryPointAssembly="SLOOB"
         EntryPointType="SLOOB.App">

   <Deployment.Parts>
   </Deployment.Parts>
   <Deployment.OutOfBrowserSettings>
   <OutOfBrowserSettings ShortName="Simple Out-of-Browser App">
    <OutOfBrowserSettings.Blurb>
     Do really cool stuff at home or on the go.
    </OutOfBrowserSettings.Blurb>
    <OutOfBrowserSettings.WindowSettings>
     <WindowSettings Title="My Cool App's Title"></WindowSettings>
    </OutOfBrowserSettings.WindowSettings>
   </OutOfBrowserSettings>
  </Deployment.OutOfBrowserSettings>
</Deployment>
```

Теперь, запустив приложение и щелкнув правой кнопкой мыши содержимое, можно увидеть, что пункт меню Install Simple Out-of-Browser App onto this computer… (Установить простое приложение, выполняющееся вне браузера, на этот компьютер) активен.

Если выбрать Install Simple Out-of-Browser App onto this computer…, выводится диалоговое окно Install application (Установка приложения), показанное на рис. 6-7.

РИС. 6-7 Сохранения для использования в автономном режиме.

Заметим, что это очень старая версия этого диалога, она может быть изменена в окончательной версии Silverlight.

В этом диалоговом окне представлены опции сохранения приложения для использования в автономном режиме либо из меню Пуск, либо с Рабочего стола. В данном случае выбран вариант размещения ярлыка в меню Пуск. Щелкните OK, чтобы сохранить приложение, или Cancel, чтобы отменить это.

Если щелкнуть OK, ярлыки будут сохранены на рабочем столе и/или в меню Пуск (в зависимости от того, что было выбрано), и приложение будет запускаться в собственном окне, как показано на рис. 6-8.

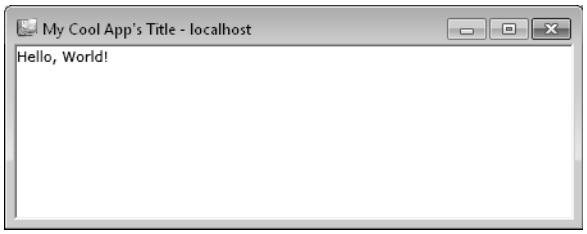

РИС. 6-8 Приложение, выполняющееся в собственном окне.

Обратите внимание, что по щелчку правой кнопкой мыши этого приложения предлагается опция Remove This Application (Удалить это приложение), которая обеспечивает удаление приложения из всех месторасположений, где оно установлено.

В этих примерах для приложения используются значки, предоставляемые Silverlight по умолчанию. Их можно переопределить с помощью PNG-файлов со значками четырех разных размеров. Таким образом, должны быть предоставлены PNG размерами 16x16, 32x32, 64x64 и 128x128, соответствующие необходимым значкам.

Понадобится добавить описание в манифест приложения, определяющее эти размеры и месторасположения файлов, и задать в Visual Studio свойству *Build Action* (Действие во время сборки) значков значение *Content* (Содержимое).

Рассмотрим манифест приложения:

```
<Deployment xmlns="http://schemas.microsoft.com/client/2007/deployment"
      xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
         EntryPointAssembly="SLOOB"
         EntryPointType="SLOOB.App">

   <Deployment.Parts>
   </Deployment.Parts>


  <Deployment.OutOfBrowserSettings>
    <OutOfBrowserSettings ShortName="Simple Out-of-Browser App">
      <OutOfBrowserSettings.Blurb>
        Do really cool stuff at home or on the go.
      </OutOfBrowserSettings.Blurb>
      <OutOfBrowserSettings.WindowSettings>
        <WindowSettings Title="My Cool App's Title"></WindowSettings>
      </OutOfBrowserSettings.WindowSettings>
      <OutOfBrowserSettings.Icons>
        <Icon Size="16x16">sl16.png</Icon>
        <Icon Size="32x32">sl32.png</Icon>
        <Icon Size="64x64">sl64.png</Icon>
        <Icon Size="128x128">sl128.png</Icon>
      </OutOfBrowserSettings.Icons>
    </OutOfBrowserSettings>

  </Deployment.OutOfBrowserSettings>
</Deployment>
```

Итак, теперь после запуска приложения и сохранения его для использования в автономном режиме, вы сможете использовать собственные значки!

Заключение

В данной главе рассматриваются возможности, предоставляемые объектом Silverlight. Мы начали с описания самого объекта, размещения его на странице и использования различных предоставляемых им параметров. Затем продемонстрировали, как с использованием JavaScript можно создать собственный мастер установки, используя возможности, предлагаемые тегом <*object*>. Наконец, вы увидели, как можно выполнять объект Silverlight вне браузера. В следующей главе будет рассмотрен механизм доступа Silverlight к объектной модели браузера, дополняющий функциональность, рассмотренную в данной главе, и предоставляющий разработчикам и дизайнерам неограниченные возможности для реализации первоклассных приложений.

Глава 7

Механизм доступа к объектной модели браузера

Мы находимся на пересечении дорог в Веб-разработке. Распри в этой области начались задолго до нас. Существует несколько лагерей, исповедующих различные философии разработки для Веб. Одни говорят, что достаточно нерасширяемого браузера, все необходимое обеспечат JavaScript, динамический HTML, каскадные таблицы стилей и AJAX. Другая сторона заявляет о том, что браузер был разработан для отображения *документов*, а не *приложений*; создавать приложения, удовлетворяющие все возрастающим требованиям пользователей, полагаясь только на браузер, все равно что строить карточный домик.

Но Microsoft Silverlight является золотой серединой для этих двух подходов. Silverlight обеспечивает мост над этой пропастью и позволяет выбирать философию. Вероятно, вы уже потратили немало времени и сил на построение API на базе JavaScript и не хотите отказываться от них и начинать все сначала, но при этом вас привлекают улучшения производительности и богатство функциональности, предлагаемые Silverlight. Тогда отбросим все страхи! Silverlight включает «доступ к браузеру», которая предоставляет разработчикам и дизайнерам неограниченные возможности для реализации первоклассных приложений. Использование механизма доступа к объектной модели браузера обеспечивает следующие возможности:

- Любой код, откомпилированный в сборку Microsoft .NET Framework, доступен браузеру и может быть вызван из JavaScript.
- Любой JavaScript-код в браузере доступен из .NET-сборки и может вызываться из Silverlight-приложения.
- Дерево визуального представления XAML доступно браузеру. С помощью JavaScript вы можете создавать XAML-элементы и добавлять их в дерево визуального представления, удалять или редактировать существующие элементы дерева визуального представления.

В данной главе обсуждаются все эти сценарии. Рассматривается API Microsoft Virtual Earth, который написан на JavaScript, и демонстрируется, насколько просто взаимодействовать с функциями JavaScript в Virtual Earth.

Создание базового приложения

Для начала нам понадобится простое приложение, предоставляющее свою функциональность браузеру и из браузера с помощью механизма доступа к объектной модели браузера.

В этом разделе мы создадим простое приложение Silverlight и затем рассмотрим обработчики, обеспечивающие возможность его использования из JavaScript.

Начнем с построения структуры .NET-приложения. Оно будет формировать визуальное представление данных для трех разных городов для каждой из трех выбранных европейских стран.

В Microsoft Visual Studio создадим новое решение Silverlight и назовем его Sample1. Убедитесь, что выбрали опцию добавления Веб-проекта в решение.

Создание класса данных

Начнем с класса, представляющего эти данные. Добавим в проект Silverlight новый класс *CityData* (Данные города).

Вот код этого класса:

```
public class CityData

{

   public string CityName { get; set; }
   public double Latitude{get;set;}
   public double Longitude{get;set;}

   public CityData(string strCityName, double nLatitude, double nLongitude)
   {
      CityName = strCityName;
      Latitude = nLatitude;
      Longitude = nLongitude;
   }
}
```

Как видите, *CityData* – очень простой класс, содержащий строку и две переменные типа double для названия города, широты и долготы его географического местоположения, соответственно.

Создание данных в Silverlight

Silverlight-приложение использует класс *CityData* для хранения данных трех городов. В реальном сценарии эти данные хранились бы в облаке, но для упрощения данные можно хранить непосредственно в приложении.

Рассмотрим метод, который возвращает данные города передаваемой в него страны:

```
private List<CityData> getCities(string strCountry)
    {
      List<CityData> ret = new List<CityData>();

      switch (strCountry)
      {
         case "france":
            {
               ret.Add(new CityData("Paris", 48.87, 2.33));
               ret.Add(new CityData("Lourdes", 43.1, 0.05));

               ret.Add(new CityData("Toulouse", 43.6, 1.38));
               break;
            }
         case "uk":
            {
               ret.Add(new CityData("London", 51.5, 0));
               ret.Add(new CityData("Stratford-Upon-Avon", 52.3, -1.71));
               ret.Add(new CityData("Edinburgh", 55.95, -3.16));
               break;
            }
         case "germany":
            {
               ret.Add(new CityData("Berlin", 52.52, 13.42));
               ret.Add(new CityData("Munich", 48.13, 11.57));
               ret.Add(new CityData("Hamburg", 53.58, 9.98));
               break;
            }
      }
      return ret;
    }
```

Этот метод довольно прост и возвращает список *List<CityData>*, встроенный в него. *List<>* создается путем создания новых экземпляров упоминаемого выше класса *CityData*, для которых в конструктор передаются название, широта и долгота.

Формирование визуального представления данных с помощью *ItemsControl*

Теперь, посмотрим, как можно использовать эти данные. Применение *List<>* для их представления обеспечивает доступность всех преимуществ функциональности связывания данных в Silverlight. Элемент управления *ItemsControl* позволяет задать в XAML представление данных, и Silverlight автоматически заполнит *ItemsControl* соответствующим числом элементов. Содержимое элементов формируется из данных соответственно заданному шаблону.

```
<UserControl x:Class="Sample1.MainPage"
   xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
   xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
   Width="400" Height="300">
   <Grid x:Name="LayoutRoot" Background="White">
      <ItemsControl x:Name="itmCities">
         <ItemsControl.ItemTemplate>
            <DataTemplate>
                  <TextBlock FontSize="14" Height="30"
                    Text="{Binding CityName}" ></TextBlock>
            </DataTemplate>
         </ItemsControl.ItemTemplate>
      </ItemsControl>
   </Grid>
</UserControl>
```

Если возникают какие-то проблемы с пониманием синтаксиса, то связывание данных рассматривается более подробно в Главе 13, «Связывание данных в Silverlight». На данный момент необходимо трактовать это следующим образом: *ItemsControl* является простым элементом управления, который всего-лишь обеспечивает желаемое представление данных. Элементы описываются с помощью *ItemTemplate* (Шаблон элемента), и, поскольку выполняется связывание данных, этот *ItemTemplate* является *DataTemplate* (Шаблон данных).

DataTemplate в Silverlight является очень мощным инструментом. Он определяет, *какие* данные необходимы и *как* они должны быть представлены.

В предыдущем примере кода определено, что визуальное представление данных будет сформировано с помощью *TextBlock* с заданным размером шрифта *FontSize="14"*. *{Binding CityName}* в поле *Text* обозначает, что выполняется привязка данных и извлечение из них поля *CityName* (Название города) для заполнения этого *TextBlock*.

Сведение воедино

Следующий шаг – получение данных. Предыдущий метод обеспечивал формирование данных. Здесь мы увидим, как попросить Silverlight сформировать их визуальное представление.

Рассмотрим метод, который создает данные для страны, загружает их в *List<CityData>* и связывает с *ItemsControl*:

```
private void upDateCities(string strCountry)
{
   List<CityData> myCities = getCities(strCountry);
   itmCities.ItemsSource = myCities;
}
```

Теперь осталось лишь вызвать этот метод. Добавим код для его вызова в конструктор MainPage() и передадим код страны ‘uk’.

```
public MainPage()
{
   InitializeComponent();
   upDateCities("uk");
}
```

Теперь приложение готово к работе. Выполним его и посмотрим на результаты. Они должны быть похожи на представленные на рис. 7-1.

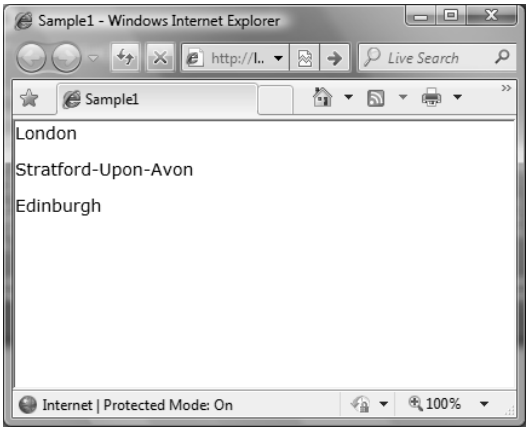

РИС. 7-1 Выполнение приложения.

На первый взгляд, не густо, но не отчаивайтесь. Это лишь база для того, что мы собираемся построить. Помните метод *upDateCities* (Обновить города)? Он вызывался из конструктора страницы и принимал значение *uk*, что обеспечивало возвращение городов Великобритании. В следующем разделе будет обеспечен доступ к этому методу из JavaScript, что позволит изменять состояние Silverlight-страницы с помощью JavaScript-кода.

Управление приложением с помощью JavaScript

В этом разделе мы продолжим дорабатывать пример, который начали в предыдущем разделе. В приложение будет добавлен некоторый HTML и JavaScript-код и в Silverlight-приложение будут введены обработчики, которые позволят JavaScript-коду управлять им.

Редактирование страницы размещения

Когда вы создали приложение Silverlight, вы добавили Веб-приложение в решение. Это решение содержит тестовую ASP.NET-страницу (под именем Sample1TestPage.aspx, если вы придерживаетесь предлагаемой здесь схемы присваивания имен) и тестовую HTML-страницу (под именем Sample1TestPage.html).

Щелкните правой кнопкой мыши страницу Sample1TestPage.html в Solution Explorer и выберите в контекстном меню *Set As Start Page* (Установить как начальную страницу). Это обеспечит, что при запуске приложения данная страница будет страницей по умолчанию.

Далее откроем эту страницу для редактирования. Внизу страницы можно видеть тег * * под именем *silverlightControlHost*. Он включает тег *<object>*, определяющий Silverlight-содержимое. На данный момент объект Silverlight не имеет идентификатора, поэтому добавляем атрибут *id="slControl"* в тег объекта, т.е. называем его *slControl*.

Перед закрывающим * * располагается *<iframe>*.

Добавим следующий код, чтобы создать на странице HTML-кнопки:

```
<div style="position: absolute; width: 400px; height: 76px;  left: 0px; top: 400px" id="JSLayer">
  <input id="bUK" type="button" value="uk"
       onclick="doCities('uk');" />
  <input id="bGermany" type="button" value="germany"
       onclick="doCities('germany');"/>
  <input id="bFrance" type="button" value="france"
       onclick="doCities('france');" />
</div>
```

Необходимо внести еще одно изменение, прежде чем запускать страницу. Каскадные таблицы стилей вверху страницы определяют, что Silverlight-содержимое должно занимать 100% экрана. Сократите размер содержимого, чтобы можно было видеть HTML-содержимое; в противном случае, Silverlight перекроет все.

Изменим код каскадных таблиц стиля, который определяет высоту и ширину элемента управления Silverlight, следующим образом:

```
#silverlightControlHost {
          height: 400px;
      width: 300px;
}
```

Теперь при выполнении приложения и загрузке HTML-страницы на экране будет отображаться примерно следующее (рис. 7-2).

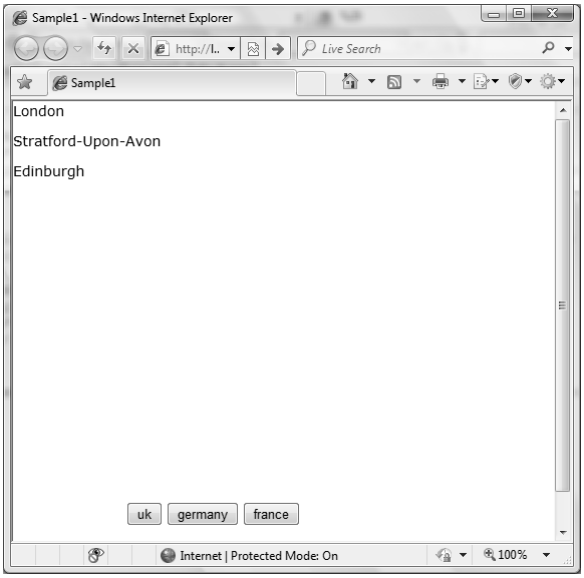

РИС. 7-2 Добавление HTML в приложение.

Написание JavaScript

Если попробовать нажать одну из кнопок, будет сформирована ошибка JavaScript, поскольку при добавлении кнопок было задано, что в случае нажатия должна выполняться JavaScript-функция *doCities* (найти города). Функция *doCities* еще не описана, сделаем это сейчас:

```
function doCities(country)
{
  var slPlugin = document.getElementById("slControl");
  slPlugin.content.MySilverlightObject.upDateCities(country);
}
```

Для получения ссылки на подключаемый модуль Silverlight функция *doCities* использует Объектную модель документов HTML (Document Object Model, DOM). Затем он вызывает метод *upDateCities*, созданную в Silverlight на .NET, для изменения состояния Silverlight-приложения и отображения новых городов.

Но погодите, ничего не работает. Во всяком случае, пока, потому что необходимо еще указать Silverlight, какие фукнции предоставлять и как это делать. Итак, возвращаемся к Silverlight-приложению.

Чтобы приложение могло использовать механизм взаимодействия с браузером, необходимо сделать две вещи.

Обеспечение поддержки сценариев в Silverlight

Первым делом, регистрируем объект Silverlight на странице как объект, поддерживающий сценарии. Это даст доступ JavaScript к методам Silverlight. Обычно это делается при первой загрузке Silverlight-приложения. Добавим обработчик события *Page_Loaded* (Страница загружена):

```
void Page_Loaded(object sender, RoutedEventArgs e)
{
   HtmlPage.RegisterScriptableObject("MySilverlightObject", this);
}
```

Это не обработчик события до тех пор, пока он не будет подключен. Сделайте это в конструкторе *MainPage()*. Вот как выглядит готовый конструктор:

```
public MainPage()
{
   InitializeComponent();
   this.Loaded += new RoutedEventHandler(Page_Loaded);
   upDateCities("uk");
}
```

Не забудьте в начале своего кода включить ссылку на *System.Windows.Browser*:

```
using System.Windows.Browser;
```

Необходимо еще одно изменение, чтобы сделать .NET-код доступным из JavaScript: методы, к которым предоставляется доступ, должны быть обозначены атрибутом *[ScriptableMember]* (Член, доступный из сценария). Это обеспечит защиту вашего кода от нежелательного зондирования JavaScript.

Метод *upDateCities* должен быть доступен из JavaScript, поэтому пометьте его этим атрибутом следующим образом:

```
[ScriptableMember]
public void upDateCities(string strCountry)
{
   myCities = getCities(strCountry);
   _cities.ItemsSource = myCities;
}
```

Обратите внимание, что все методы, обозначенные как поддерживающие сценарии, должны быть *public* методами. Теперь ваше приложение полностью готово к совместной работе с JavaScript. Для вызова используется следующая JavaScript-функция:

```
slPlugin.content.MySilverlightObject.upDateCities(country);
```

Элемент управления Silverlight предоставляет свое содержимое через *<control>.content collection*. Объект, поддерживающий сценарии, зарегистрирован под именем *MySilverlightObject*, поэтому он присутствует, и его предоставляемые методы доступны для вызова.

Сведение воедино

Теперь при выполнении приложения нажатие кнопок обеспечивает передачу названия страны в Silverlight. Это значение используется для формирования списка *List<CityData>*, с которым связан

пользовательский интерфейс. Это связывание обеспечивает вывод новых названий городов в окне Silverlight-приложения.

Это можно увидеть на рис. 7-3. В данном примере была выбрана Германия.

Не так уж сложно, правда? Мы просто предоставили возможность управления Silverlight-приложением из браузера. То есть, как видите, теперь есть возможность улучшить существующие сайты и JavaScript-код уровнем более насыщенного визуального представления, который обеспечивает Silverlight.

Но не только ваш *код* доступен Silverlight. XAML, генерируемый Silverlight (с использованием дерева визуального представления), может напрямую изменяться извне Silverlight-приложения. Рассмотрим это в следующем разделе.

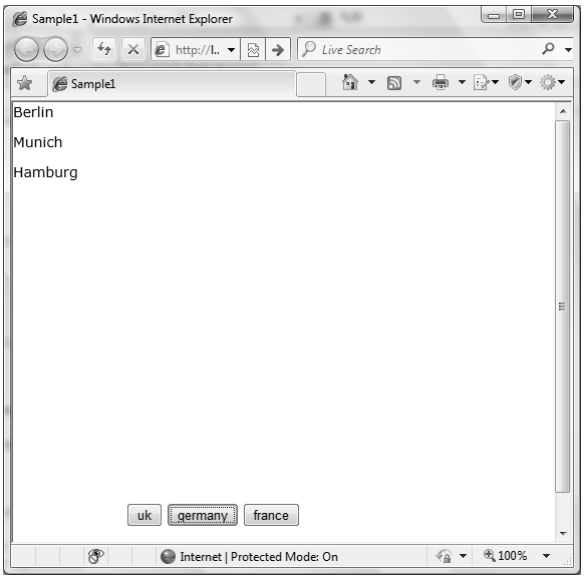

РИС. 7-3 Изменение Silverlight-содержимого с помощью JavaScript.

Изменение дерева визуального представления Silverlight

В предыдущем разделе было показано, как можно сделать свой .NET-код доступным из JavaScript. Кроме этого, JavaScript можно использовать для редактирования дерева визуального представления Silverlight через добавление, удаление или редактирование XAML.

В данном разделе рассмотрим, как реализуется изменение дерева визуального представления, и добавим следующую функциональность: если пользователь нажимает кнопку страны, и надписи с названием этой страны не существует, Silverlight-приложение создает новую надпись.

После создания надписи с названием страны для ее отображения необходимо изменить пользовательский интерфейс (UI). В этом случае требуется сместить данные вниз, чтобы освободить место для надписи.

Если надпись с названием страны уже существует, по нажатию одной из кнопок пользователем ее можно редактировать, используя название страны. Как это выглядит, показано на рис. 7-4.

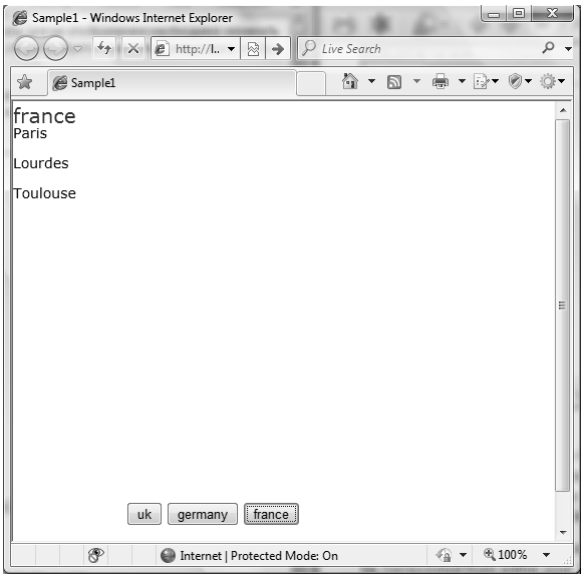

РИС. 7-4 *Изменение XAML-дерева.*

Для удобства повторим здесь код функции *doCities*, реализующий обработку нажатий кнопок:

```
function doCities(country)
{
  var slPlugin = document.getElementById("slControl");
  slPlugin.content.MySilverlightObject.upDateCities(country);
}
```

Как видите, в первой строке получаем ссылку на элемент управления Silverlight и сохраняем ее в переменную *slPlugin*.

На рис. 7-4 можно увидеть большой заголовок вверху экрана. Эта надпись формируется XAML-элементом *titleText* (Заголовок), поэтому, прежде всего, необходимо проверить, существует ли надпись. Если существует, просто редактируем ее, используя переданное при нажатии кнопки название страны.

Это можно сделать с помощью метода *findName* элемента управления Silverlight. Рассмотрим код:

```
var titleExists = slPlugin.content.findName("titleText");
if (titleExists) {
   slPlugin.content.findName("titleText").Text = country;
}
```

Но *titleText* может и не существовать (это возможно при первом выполнении приложения). Вернитесь к XAML-коду, описывающему пользовательский интерфейс, и вы не найдете там *TextBlock* с именем *titleText*. В представленный выше *if* необходимо добавить выражение *else* для создания *TextBlock*.

Давайте строка за строкой рассмотрим, как это делается.

Этапы создания XAML-элемента просты. Сначала создается JavaScript-переменная, включающая строку, которая определяет XAML. Затем из этого в элементе управления Silverlight создается XAML-элемент. Наконец, XAML-элемент добавляется или вставляется в дерево визуального представления.

Итак, шаг 1, создание XAML:

```
var xamlFragment = '<TextBlock FontSize="20" Foreground="Blue">' + country + '</TextBlock>';
```

Этот XAML не имеет имени, а нам необходимо назвать его *titleText*, поэтому требуется еще кое-что добавить. Как вы, вероятно, помните, для присваивания имен элементу в XAML применяется синтаксис *x:Name*, который использует пространство имен. Итак, требуется добавить не только тег, но также и объявление пространства имен:

```
var xamlFragment = '<TextBlock  '
   +' xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml" '
   +' x:Name="titleText" FontSize="20" Foreground="Blue">' + country + '</TextBlock>';
```

В шаге 2 из этой строки создаем XAML-элемент в элементе управления Silverlight. Это можно реализовать методом *createFromXaml* следующим образом:

```
var tb = slPlugin.content.createFromXaml(xamlFragment, false);
```

Это не обеспечивает формирования визуального представления XAML, здесь просто создаются объекты, и Silverlight получает возможность формировать их представление. Для формирования визуального представления этих объектов они должны быть добавлены в дерево визуального представления, это делается в шаге 3.

При добавлении или вставке объекта в дерево визуального представления необходимо учесть несколько вещей. Во-первых, *куда* должно быть вставлено содержимое. Оно должно быть добавлено как дочерний элемент текущего элемента, поэтому сначала необходимо определиться с тем, какой элемент должен стать родительским элементом данного объекта. При использовании метода *Add* (Добавить) содержимое будет добавлено как последний дочерний элемент текущего элемента. Если используется метод *Insert* (Вставить), содержимое будет добавлено в заданное местоположение.

В XAML-коде для этого приложения имеется <*Grid*> с корневым элементом *LayoutRoot*. Поместите новое содержимое в корневой элемент. Сделайте его первым дочерним элементом, чтобы он отображался над существующим содержимым.

Рассмотрим код:

```
slPlugin.content.findName("LayoutRoot").children.insert(0, tb);
```

Теперь сведем все вместе, чтобы посмотреть, как все будет выглядеть:

```
<script type="text/javascript">
   function doCities(country) {
      var slPlugin = document.getElementById("slControl");
      var titleExists = slPlugin.content.findName("titleText");
      if (titleExists) {
         slPlugin.content.findName("titleText").Text = country;
      }
      else {
         var xamlFragment = '<TextBlock '
           + 'xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml" '
           + 'x:Name="titleText" FontSize="20" '
           + 'Foreground="Blue">' + country + '</TextBlock>';
         var tb = slPlugin.content.createFromXaml(xamlFragment, false);
         slPlugin.content.findName("LayoutRoot").children.insert(0, tb);

      }

      slPlugin.content.MySilverlightObject.upDateCities(country);
   }
</script>
```

Запуская приложение вы надеетесь увидеть что-то, подобное представленному ранее на рис. 7-4. Ну, попробуйте. На самом деле получаем следующее (рис. 7-5).

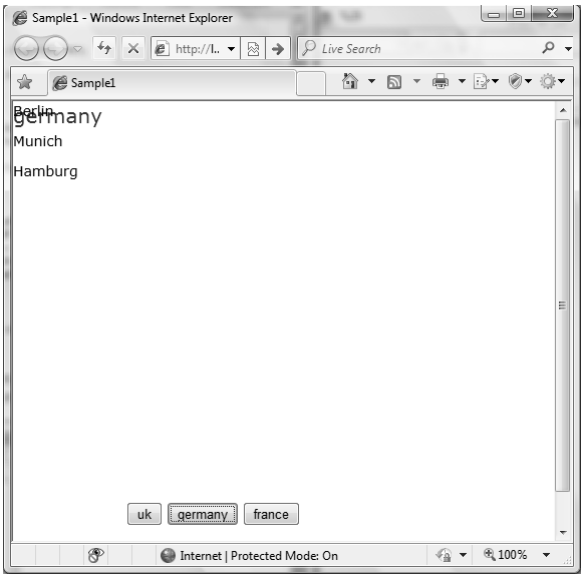

РИС. 7-5 Выполнение нового приложения.

Как видно на рис. 7-5, новый блок заголовка отображается поверх остального текста, несмотря на то, что был вставлен в положение 0, т.е. должен отображаться выше остального содержимого. Почему так получилось? Причина в том, что визуальное представление содержимого формируется в <*Grid*>, поэтому оно не позиционируется автоматически. Если бы в качестве контейнера использовался элемент, обеспечивающий автоматическую компоновку, такой как <*StackPanel*>, например, блок заголовка отображался бы правильно. Но применяемый контейнер не обеспечивает компоновки.

Содержимое представлено с помощью элемента *itmCities* типа <*ItemsControl*>. Это дочерний элемент *Grid*. Позиционировать содержимое в сетке можно с помощью свойства *Margin*.

Далее представлен JavaScript-код, в котором задается свойство *Margin*, что обеспечивает смещение содержимого вниз на несколько пикселов:

```
slPlugin.content.findName("itmCities").SetValue("Margin", "0,22,0,0");
```

Добавьте этот код в конец выражения *else* в JavaScript, чтобы он выполнялся только один раз, при создании заголовка.

Теперь при выполнении приложения получаем результат, представленный на рис. 7-6. Именно такое представление и требовалось.

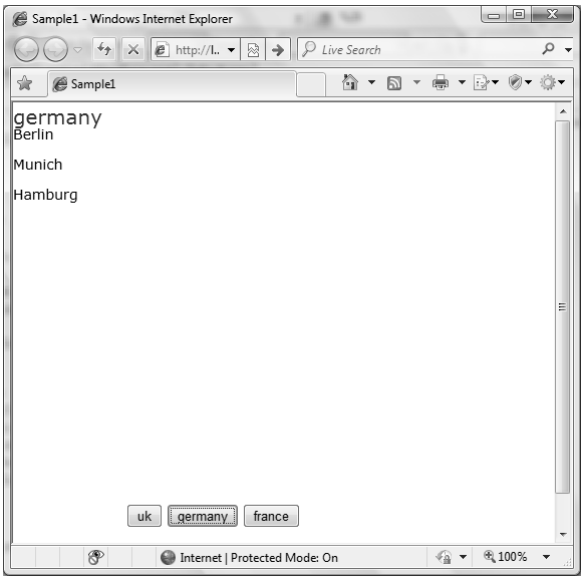

РИС. 7-6 Выполнение приложения.

Хотя, кажется, слишком много хлопот ради такого простого эффекта, но подумайте о смысле только что сделанного. Вы радикально изменили функциональность и представление выполняющегося Silverlight-приложения без перекомпиляции кода. Сделано это за счет использования JavaScript-кода и механизма доступа к браузеру!

В следующем разделе будет рассмотрена реализация возможности управления JavaScript-кодом и HTML-содержимым из .NET-кода в Silverlight.

Доступ к функциям JavaScript из .NET

Данный раздел замыкает цикл вопросов, обсуждаемых до сих пор. Мы рассмотрим использование .NET-кода для взаимодействия с функциями JavaScript на странице. В этом есть практическая польза, поскольку это позволит продолжать применять существующие API JavaScript, и их не придется переписывать в Silverlight.

Например, платформа Microsoft Virtual Earth может использоваться для построения приложений с использованием карты. В этом разделе будем работать с этим API.

Для данного примера продолжим редактировать HTML-страницу. Чтобы использовать Virtual Earth, необходимо сделать следующее.

Сначала добавим в страницу тег <*div*>, в котором будет располагаться карта.

Найдите на странице элемент <*div*>, содержащим тег <*object*>, и поместите новый <*div*> сразу над ним:

```
<div id='mapDIV' style="position: absolute; width: 443px; height: 417px; z-index: 2; left: 301px; top: 0px"></div>
```

Далее добавим ссылку на библиотеки JavaScript для Virtual Earth. Их можно добавить в любом месте раздела <*head*> страницы:

```
<script type="text/javascript" src="http://dev.virtualearth.net/mapcontrol/mapcontrol.ashx?v=6.2"></script>
```

Затем напишем JavaScript-функцию, которая будет загружать элемент управления *Map* (Карта) в созданный *div*:

```
var map = null;

function GetMap()
{
   map = new VEMap('mapDIV');
   map.LoadMap();
}
```

Данный код определеяет переменную *map*, которая будет использоваться совместно данной и еще одной функцией JavaScript (которая будет продемонстрирована через мгновение). *GetMap()* создает в созданном ранее *div* новый объект *VEMap* (представляющий карту Virtual Earth) и загружает его.

Наконец, вызываем *GetMap* при загрузке страницы. Скорректируйте тег <*Body*> и определите его поведение *OnLoad*, чтобы обеспечить вызов *GetMap*:

```
<body onload="GetMap();">
```

После выполнения этой страницы на экране можно будет увидеть загруженную карту по умолчанию, как показно на рис. 7-7.

Чтобы можно было работать с этой картой из .NET-кода, необходимо добавить JavaScript-функцию, которая будет находить необходимое местоположение и перемещать карту соответственно ему. Эта функция может вызываться из другого JavaScript или .NET-кода.

Вот эта функция:

```
function MoveMap(where) {
   try {
      map.Find(null, where);
   }
   catch (e) {
      alert(e.message);
   }

}
```

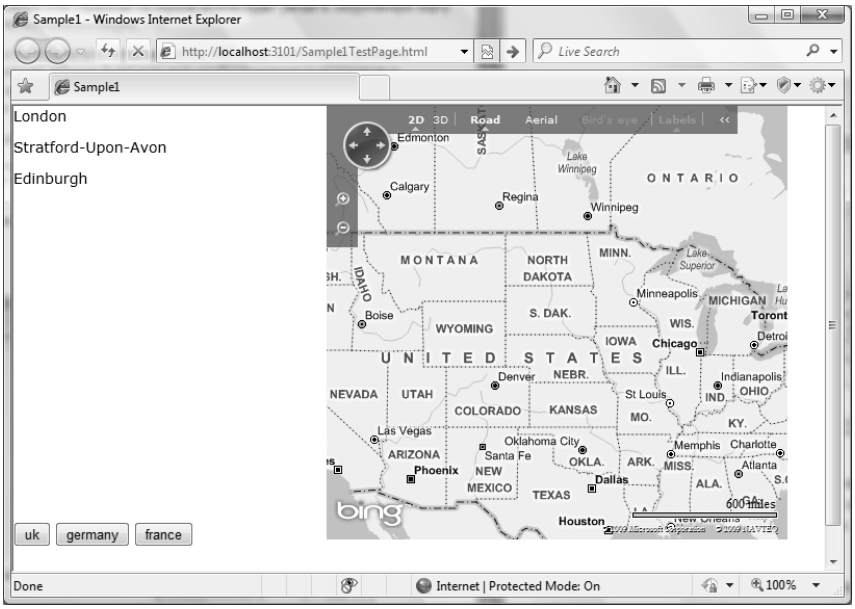

РИС. 7-7 Включение карты Virtual Earth.

Переменная *map* создается ранее при загрузке карты Virtual Earth. Данная функция просто использует ее API *Find* (Найти) и передает в него строку *where* (где). Как видите, *where* – это параметр, передаваемый в эту функцию. Поэтому, такой вызов:

```
MoveMap("London,England");
```

обеспечит перемещение карты в указанное местоположение.

В заключение, рассмотрим .NET-код, реализующий функциональность перемещения карты в заданное местоположение, когда пользователь щелкает одно из них.

Первым делом, необходимо обеспечить возможность нажатия точек данных. Вспомним, что ранее при описании XAML для размещения данных использовался элемент <*ItemsControl*>. Этот элемент имеет шаблон <*DataTemplate*>, определяющий представление данных, и он включает простой <*TextBlock*> для каждого элемента управления. Таким образом, чтобы сделать *все* элементы активными, необходимо просто сделать активизируемым <*TextBlock*> в <*DataTemplate*>. Это можно реализовать посредством обработки события *MouseLeftButtonUp*:

```
<ItemsControl x:Name="itmCities">
   <ItemsControl.ItemTemplate>
      <DataTemplate>
            <TextBlock FontSize="14" Height="30"
               Text="{Binding CityName}"
               MouseLeftButtonUp="TextBlock_MouseLeftButtonUp" >
              </TextBlock>
      </DataTemplate>
   </ItemsControl.ItemTemplate>
</ItemsControl>
```

Теперь по щелчку *любого TextBlock* вызывается метод *TextBlock_MouseLeftButtonUp*.

Этот метод должен принимать город и страну, создавать строку, их содержащую, и вызывать JavaScript-функцию *MoveMap* (Передвинуть карту) для обновления местоположения на карте.

Сначала создадим две строки для города и страны:

```
String strCity = "";
String strCountry = "";
```

В обработчиках событий Silverlight объект, формирующий событие, передается как первый параметр. Чтобы получить данные города, просто выполняем приведение к *TextBlock* и принимаем его параметр *Text*:

```
TextBlock clickedText = sender as TextBlock;
strCity = clickedText.Text;
```

С данными страны немного сложнее, потому что содержащий их *TextBlock* добавляется в дерево визуального представления во время выполнения. (Об этом подробно рассказывается в предыдущем разделе.) Найти страну в дереве визуального представления поможет следующий код:

```
TextBlock title = this.FindName("titleText") as TextBlock;
```

Данные страны добавляются в дерево визуального представления только при первом щелчке пользователя одной из кнопок с названиями стран. Поэтому, прежде чем извлекать из текста заголовка данные страны, необходимо проверить, не равен ли он *null*, поскольку в этом случае будет сформирована ошибка времени выполнения.

```
if (title != null)
{
   strCountry = title.Text;
}
```

Получив данные города и страны, формируем из них строку:

```
string toFind = strCity + "," + strCountry;
```

Вот здесь происходит волшебство. Необходимо вызвать JavaScript-функцию и передать ей эту строку в качестве параметра.

С помощью коллекции *HtmlPage* в .NET можно найти JavaScript-функцию как свойство страницы. Поскольку это сценарий, необходимо привести его к объекту сценария. Рассмотрим код:

```
ScriptObject sMoveMap = (ScriptObject)HtmlPage.Window.GetProperty("MoveMap");
```

Теперь, осталось лишь выполнить сценарий и передать в него строку *toFind*. Это можно сделать с помощью метода *InvokeSelf* объекта *ScriptObject*:

```
sMoveMap.InvokeSelf(toFind);
```

Вот как выглядит полный обработчик события:

```
private void TextBlock_MouseLeftButtonUp(object sender, MouseButtonEventArgs e)
{
  String strCity = "";
  String strCountry = "";
  TextBlock clickedText = sender as TextBlock;
  strCity = clickedText.Text;

  TextBlock title = this.FindName("titleText") as TextBlock;
  if (title != null)
  {
    strCountry = title.Text;
  }

  string toFind = strCity + "," + strCountry;

ScriptObject sMoveMap = (ScriptObject)HtmlPage.Window.GetProperty("MoveMap");
```

```
  sMoveMap.InvokeSelf(toFind);
}
```

Это все что необходимо для вызова JavaScript-функций из .NET.

На рис. 7-8 представлены результаты, получаемые по щелчку Гамбург, Германия; и на рис. 7-9 – по щелчку Париж, Франция.

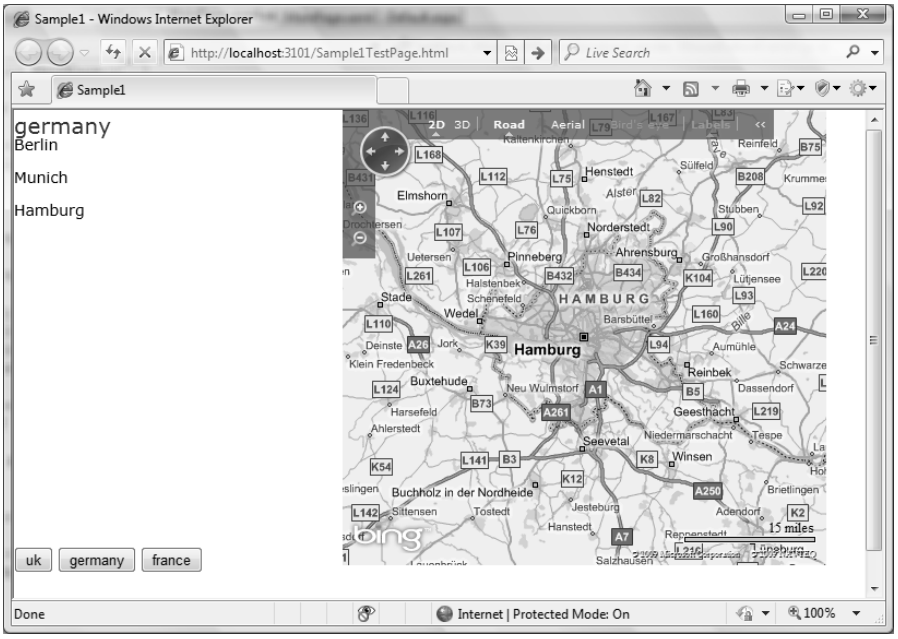

РИС. 7-8 Использование UI для перемещения карты в Гамбург, Германия.

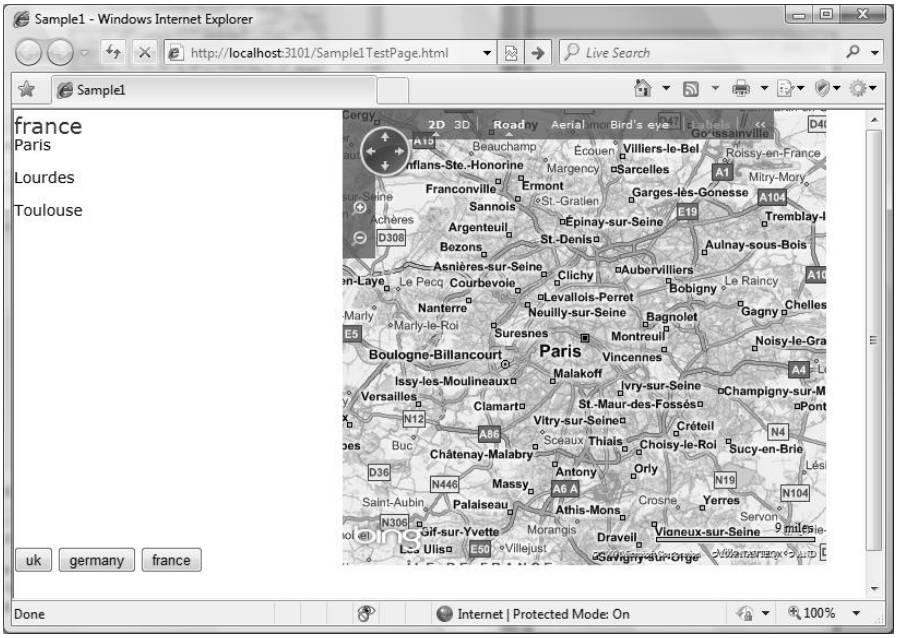

РИС. 7-9 Использование UI для перемещения карты в Париж, Франция.

Это всего лишь краткий обзор механизмов взаимодействия с браузером. Он иллюстрирует гибкость и мощь Silverlight и то, почему Silverlight позволяет создавать первоклассные приложения для браузера!

Заключение

Механизм доступа к объектной модели браузера – замечательная технология, позволяющая интегрировать Silverlight в существующие Веб-приложения. Возможны разнообразные варианты. С его помощью можно вызывать .NET-код и использовать .NET-код из JavaScript, изменять визуальное

представление Silverlight из HTML-страницы и вызывать существующие функции JavaScript из .NET-кода.

В этой главе шаг за шагом было создано приложение, реализующее сценарий, уже реализованный кем-то в API JavaScript, а именно, Microsoft Virtual Earth. Это пример демонстрирует, как можно добавлять Silverlight-содержимое в приложение, использующее эти API. В следующей главе немножко переключим наше внимание и рассмотрим основные элементы управления XAML, доступные в Silverlight.

Глава 8

Основные элементы управления Silverlight

Пространство имен *System.Windows* включает ряд основных элементов управления, составляющих сердце каждого приложения Microsoft Silverlight. В данной главе представлены и рассматривается использование следующих элементов управления:

- *Button*
- *CheckBox*
- ComboBox
- *HyperlinkButton*
- *Image*
- *ListBox*
- *RadioButton*
- *TextBlock*
- *TextBox* и *PasswordBox*

Будут рассмотрены специальные свойства, методы и события, предоставляемые каждым из этих элементов управления, а также свойства, методы и события, общие для всех элементов управления. Изложенная в данной главе информация не должна быть единственным источником при работе с любым из данных элементов управления, но здесь приводится достаточно базовых сведений, чтобы подтолкнуть вас и помочь начать самостоятельные эксперименты с ними.

Элемент управления *Button*

В Silverlight кнопка реализуется с помощью элемента управления *Button*. Кнопка реагирует на ввод пользователя с таких устройств ввода, как мышь, клавиатура или стилус, формируя событие *Click* (Щелчок). Условия формирования кнопкой события *Click* могут быть настроены несколькими способами. Они задаются свойством *ClickMode* (Режим щелчка), которое может принимать значения *Hover* (Наведение), *Press* (Нажатие) и *Release* (Отпустить). Эти значения определяют момент формирования события *Click*. В первом случае, *Hover*, событие *Click* формируется при наведении курсора мыши на кнопку. Во втором случае, *Press,* оно формируется при нажатии кнопки мыши в момент, когда курсор находится на *Button*. В третьем случае, *Release*, это событие возникает, когда пользователь нажимает *и* отпускает кнопку мыши при наведенном на кнопку курсоре.

```
<Canvas x:Name="LayoutRoot" Background="White">
  <Button x:Name="b1" ClickMode="Hover"
     Content="Button1" Click="Button_Click"></Button>
  <Button x:Name="b2" Canvas.Top="40" ClickMode="Press"
     Content="Button2" Click="Button_Click"></Button>
  <Button x:Name="b3" Canvas.Top="80" ClickMode="Release"
     Content="Button3" Click="Button_Click"></Button>
</Canvas>
```

Как видите, для всех этих кнопок определен обработчик события *Click* и одна и та же функция, *Button_Click*, для каждой из них.

Вот код обработки этого события на языке C#:

```
private void Button_Click(object sender, RoutedEventArgs e)
{
  Button b = (Button)sender;
  string strTest = b.Name;
}
```

Этот код построен по типичному шаблону обработчика события. Он принимает объект отправителя, содержащий ссылку на элемент управления, который сформировал событие (в данном случае, одну из кнопок), и набор аргументов, связанных с этим событием (*RoutedEventArgs*), который включает метаданные о событии.

Вы заметите, что в данном экземпляре объявлен только один обработчик события. Это делает код аккуратным и понятным. Конечно, вы захотите выяснить, какой из элементов сформировал событие, для этого пригодится отправитель.

Отправитель – это объект типа *Object*, поэтому чтобы извлечь из него характерные для кнопок свойства, его просто приводят к типу , *Button*. В данном случае, после приведения этого объекта к типу *Button* мы можем получить его имя. Если запустить приложение сейчас, можно будет увидеть три кнопки и то, как свойство *ClickMode*, ассоциированное с каждой из этих кнопок, обусловливает их разное поведение!

Дополнительным очень полезным свойством является *IsEnabled* (Активирован). Если *IsEnabled* задано значение *false*, кнопка отображается на экране, но выглядит неактивной и не формирует события.

Надпись на кнопке задается свойством *Content* (Содержимое). В качестве его значения может быть задан простой текст, который будет отображаться как название кнопки. Однако *Button* также является контейнером, поэтому ее содержимое можно настроить с помощью Extensible Application Markup Language (XAML).

Вот пример кнопки (ужасно некрасивой), содержимое которой описано с помощью XAML:

```
<Button x:Name="b1" Click="Button_Click" Width="100" Height="100">
  <Canvas>
    <Ellipse Fill="Green" Width="50" Height="50"></Ellipse>
    <TextBlock Text="Hello"></TextBlock>
  </Canvas>
</Button>
```

Как видите, элемент управления *Button* обеспечивает чрезвычайную гибкость в представлении кнопок, что позволяет создавать разнообразные кнопки в Веб-приложениях, например, такие как можно увидеть на панели инструментов Microsoft Office 2007.

Элемент управления *CheckBox*

Элемент управления *CheckBox* предоставляет пользователю опцию для выбора. Обычно он представляется в виде прямоугольника, в котором пользователь может установить или убрать флажок. Он используется, когда имеется ряд опций на выбор и можно выбрать несколько из них.

При установлении флажка *CheckBox* формирует событие *Checked* (Установлен), при снятии флажка формируется событие *Unchecked* (Снят), и событие *Click* формируется при каждом щелчке по *CheckBox*. Аналогично элементу управления *Button*, *CheckBox* имеет свойство *ClickMode*, которому может быть задано значение *Hover, Press* или *Release,* определяющее условия формирования события.

CheckBox может быть флажком с тремя состояниями (поддерживается неопределенное состояние). Это определяется свойством *IsThreeState* (Три состояния). Если это состояние включено, и пользователь переводит *CheckBox* в неопределенное состояние, свойству *IsChecked* присваивается значение *null*.

Значение *CheckBox* можно получить с помощью свойства *IsChecked*. Необходимо быть осторожным с *IsChecked*, когда *CheckBox* находится в режиме с тремя состояниями, поскольку, если *CheckBox* в неопределенном состоянии, свойство *IsChecked* возвращает значение *null*.

Аналогично элементу управления *Button*, *CheckBox* является контейнером. Он имеет свойство *Content*, с помощью которого можно задавать простую строку текста, которая будет отображаться как надпись рядом с флажком, или можно разместить XAML-код в дочернем теге <*Content*>, для получения более насыщенного эффекта.

Вот пример каждого из вариантов текста:

```
<StackPanel>
  <CheckBox Checked="CheckBox_Checked"
     Unchecked="CheckBox_Unchecked"
     IsThreeState="True" Content="Test1">
  </CheckBox>
  <CheckBox Checked="CheckBox_Checked"
     Unchecked="CheckBox_Unchecked"
     IsThreeState="True">
    <CheckBox.Content>
      <StackPanel Orientation="Horizontal">
        <TextBlock Text="The Caption"></TextBlock>
        <Image Source="..."/>
      </StackPanel>
    </CheckBox.Content>
  </CheckBox>
</StackPanel>
```

При обработке событий *Clicked, Checked* и *Unchecked*, в качестве *sender* будет выступать объект, который необходимо привести к типу *CheckBox*, чтобы получить доступ к свойствам флажка.

Ниже приведен пример:

```
private void CheckBox_Checked(object sender, RoutedEventArgs e)
{
  CheckBox c = (CheckBox)sender;
  bool b = (bool)c.IsChecked;
}

private void CheckBox_Unchecked(object sender, RoutedEventArgs e)
{
  CheckBox c = (CheckBox)sender;
  bool b = (bool)c.IsChecked;
}
```

Элемент управления *ComboBox*

Элемент управления *ComboBox* (Поле со списком) используется для представления списка доступных для выбора элементов. При этом отображается только выбранный элемент, но имеется выпадающее меню с полным списком доступных для выбора элементов.

Список образуют объекты *ComboBoxItem* (Элемент поля со списком), как в данном примере:

```
<ComboBox>
  <ComboBoxItem>
    <TextBlock Text="Item1"></TextBlock>
  </ComboBoxItem>
  <ComboBoxItem>
    <TextBlock Text="Item2"></TextBlock>
  </ComboBoxItem>
  <ComboBoxItem>
    <TextBlock Text="Item3"></TextBlock>
  </ComboBoxItem>
 </ComboBox>
```

Обработка выбора элемента *ComboBox* пользователем осуществляется посредством события *SelectionChanged* (Выбранный элемент изменен) и свойства *SelectedItem* (Выбранный элемент).

Эти события объявляются в <*ComboBox*>, а не его элементах, так что *SelectionChanged* используется следующим образом:

```
<ComboBox SelectionChanged="ComboBox_SelectionChanged">
 ...
</ComboBox>
```

В данном коде объявляется обработчик события изменения выбранного элемента *ListBox_SelectionChanged*. Это событие обрабатывается путем приведения *SelectedItem* к соответствующему типу с последующим чтением его свойств. В предыдущем примере элементы *ComboBox* содержали элементы управления *TextBlock*, поэтому получить их содержимое можно следующим образом:

```
private void ComboBox_SelectionChanged(object sender, SelectionChangedEventArgs e)
{
  ComboBox theBox = sender as ComboBox;
  ComboBoxItem theItem = theBox.SelectedItem as ComboBoxItem;
  TextBlock theTextBlock = theItem.Content as TextBlock;
}
```

Обратите внимание, что *SelectedItem* из *ComboBox* возвращает *объект*, а не *ComboBoxItem*, потому что *ComboBox* может размещать различные типы содержимого в *ComboBoxItem*. В данном случае, чтобы использовать *SelectedItem,* приводим его к *ComboBoxItem*.

Продемонстрируем вариант, когда *ComboBox* содержит элементы разного типа. Ниже представлен пример, в котором *ComboBox* включает составные элементы, образованные компонентами *StackPanel*, в каждом из которых имеется *Rectangle*, *Image* и *TextBlock*:

```
<ComboBox x:Name="theList" SelectionChanged="theList_SelectionChanged">
  <ComboBoxItem>
    <StackPanel Orientation="Horizontal">
      <Rectangle Fill="Black" Height="100" Width="100"></Rectangle>
      <Image Height="100" Width="100" Source="sl.jpg"/>
      <TextBlock Text="Item 1"></TextBlock>
    </StackPanel>
    </ComboBoxItem>
    <ComboBoxItem>
      <StackPanel Orientation="Horizontal">
        <Rectangle Fill="Black" Height="100" Width="100"></Rectangle>
        <Image Height="100" Width="100" Source="sl.jpg"/>
        <TextBlock Text="Item 2"></TextBlock>
      </StackPanel>
    </ComboBoxItem>
  </ComboBox>
```

Данный фрагмент кода обеспечивает формирование более насыщенного визуального представления элементов *ComboBox* (включающего прямоугольник, изображение и текстовый блок). Необходимое содержимое можно извлечь в обработчике события, приводя соответствующий дочерний элемент к соответствующему значению. Например:

```
ComboBoxItem item = theList.SelectedItem as ComboBoxItem;
StackPanel s = item.Content as StackPanel;
TextBlock t = s.Children[2] as TextBlock;
string strValue = t.Text;
```

Заметьте, что в данном примере *ComboBox* назван *theList*, поэтому нет необходимости приводить параметр *sender*.

Еще один подобный пример, который также демонстрирует возможности формирования визуального представления гибкого содержимого элементов управления содержимым Silverlight, в разделе «Элемент управления *ListBox*» далее в этой главе.

Элемент управления *HyperlinkButton*

HyperlinkButton (Гиперссылка) обеспечивает на странице элемент с возможностью щелчка по нему для перехода по URI, заданному свойством *NavigateUri* (Uri перехода).

Например:

```
<HyperlinkButton Content="Microsoft"
   NavigateUri="http://www.microsoft.com" />
```

HyperlinkButton – это элемент управления с содержимым, т.е. он может иметь простое свойство *Content*, содержащее текст, который будет отображаться для гиперссылки, или его содержимое может быть намного более насыщенным и более привлекательным.

Например, если вы хотите, чтобы гиперссылка выглядела как рисунок, это очень просто реализовать, используя дочерний элемент <*Hyperlink.Content*>, в котором может располагаться изображение. Например:

```
<HyperlinkButton NavigateUri="http://www.silverlight.net" >
  <HyperlinkButton.Content>
    <Image Source="sl.jpg"/>
  </HyperlinkButton.Content>
</HyperlinkButton>
```

Теперь, когда приложение будет запущено, вы получите изображение, щелчок по которому обеспечит переход по заданному URL.

Свойство *TargetName* определяет, где будет открываться содержимое, располагающееся по заданному URL. Для вывода на экран содержимого URL доступны следующие опции:

- **_blank** Открывает новое окно браузера без имени.
- **_self** Замещает текущую HTML-станицу новым содержимым. Если HTML-станица отображается во фрейме, замещается только содержимое рамки.
- **_top** Замещает текущую HTML-станицу новым содержимым. Даже если HTML-станица отображается во фрейме, меняется содержимое всего окна браузера.
- **_parent** Замещает всю HTML-станицу.

Если используется фреймовая структура, можно задавать имя используемого фрейма.

Вот пример, определяющий, что когда пользователь выбирает этот элемент управления, должно открываться новое окно браузера:

```
<Grid x:Name="LayoutRoot" Background="White">
  <HyperlinkButton NavigateUri="http://www.silverlight.net"
       TargetName="_blank" >
    <HyperlinkButton.Content>
      <Image Source="sl.jpg"/>
    </HyperlinkButton.Content>
  </HyperlinkButton>
</Grid>
```

Обратите внимание, что событие *Click* для *HyperlinkButton* формируется перед переходом по заданному URI. Это может быть очень полезным, потому что обеспечивает завершение любой предварительной обработки перед загрузкой нового содержимого.

Далее рассмотрим пример с использованием события *Click*. Обратите внимание, что *NavigateUri* задано значение *http://www.microsoft.com*, но фактически страница переходит по адресу *http://www.silverlight.net*, потому что *Click* выполняется раньше перехода и меняет значение *NavigateUri*.

Во-первых, рассмотрим XAML:

```
<HyperlinkButton NavigateUri="http://www.microsoft.com"
   TargetName="_blank" Click="HyperlinkButton_Click">
  <HyperlinkButton.Content>
    <Image Source="sl.jpg"/>
  </HyperlinkButton.Content>
</HyperlinkButton>
```

И теперь обратим внимание на выделенный C#-код, в котором обрабатывается событие *HyperlinkButton_Click*:

```
private void HyperlinkButton_Click(object sender, RoutedEventArgs e)
{
  HyperlinkButton h = (HyperlinkButton)sender;
  string strTest = h.NavigateUri.AbsoluteUri;
  if (strTest == "http://www.microsoft.com/")
  {
    h.NavigateUri = new Uri("http://www.silverlight.net");
  }
}
```

При выполнении этого кода вы увидите, что несмотря на то, что для *HyperlinkButton* определен один URL, в ходе обработки события *Click* ему будет задан другой URL, и переход будет выполнен по новому URL.

Элемент управления *Image*

Элемент управления *Image* используется для отображения изображений. Он может принимать форматы файлов .bmp, .jpg и .png. Путь к изображению задается свойством *Source*. Например:

```
<Image Source="sl.jpg" />
```

> **Примечание** Если в коде дается ссылка на неподдерживаемый формат, или *Source* задан несоответствующим образом, будет сформировано событие *ImageFailed* (Ошибка изображения).

Если фактические размеры изображения отличаются от заданных для элемента управления *Image* (т.е. если имеется элемент *Image* размером 100 × 100 и в него загружается .jpg-изображение 2000 × 2000 пикселов), управлять поведением отображения можно с помощью свойства *Stretch*. Оно может принимать следующие значения:

- ***Fill*** Изменяет размеры изображения соответственно размерам области вывода, используя независимое масштабирование по осям x и y.
- ***Uniform*** Изменяет размеры изображения так, чтобы оно поместилось в элемент управления *Image*, но при этом сохраняет пропорции изображения.
- ***UniformToFill*** Изменяет размеры изображения так, чтобы оно полностью заполнило область вывода, обрезая его в случае необходимости.
- ***None*** Отображает изображение как есть, что приводит к его обрезке, если оно не помещается в элемент *Image*.

Часто изображение необходимо задать не во время разработки, указывая URI в XAML-коде, а определить его во время выполнения, возможно, как результат обращения к базе данных или чего-либо подобного. В таких случаях в качестве источника для элемента управления *Image* может использоваться класс *BitmapImage* (пространства имен *System.Windows.Media.Imaging*). Далее приведен пример того, как это делается.

Сначала XAML:

```
<Image x:Name="theImage"/>
```

Затем в файле выделенного C# кода необходимо подключить пространство имен:

```
using System.Windows.Media.Imaging;
```

И вот код, который обеспечит загрузку изображения по URL при загрузке страницы:

```
public Page(){
  InitializeComponent();
  this.Loaded += new RoutedEventHandler(Page_Loaded);
}

void Page_Loaded(object sender, RoutedEventArgs e)
{
  Uri uri = new Uri("sl.jpg",UriKind.Relative);
  theImage.Source = new BitmapImage(uri);
}
```

В данном случае создается новый объект *Uri* с использованием пути к изображению. Второй параметр в конструкторе – опция, определяющая, как должен вычисляться Uri. Используемый URI может быть относительным (в этом случае поиск заданного ресурса будет вестись в том же каталоге, в котором располагается компонент), абсолютным (в этом случае поиск заданного ресурса будет вестись напрямую, поэтому местоположение должно указываться с применением синтаксиса *http://сервер/ресурс*) или комбинацией первых двух.

Когда Uri создан, он может использоваться для создания нового *BitmapImage*, который определяется как источник для элемента управления *Image*.

Элемент управления *ListBox*

Элемент управления *ListBox* используется для представления содержимого в виде упорядоченного списка. Это достаточно гибкий элемент управления, так что элементы списка могут создаваться из содержимого любого типа, но обычно список образуется элементами *ListBoxItem* (Элемент списка), как показано в следующем примере:

```
<ListBox x:Name="theList" SelectionChanged="ListBox_SelectionChanged">
  <ListBoxItem Content="1"/>
  <ListBoxItem Content="2"/>
  <ListBoxItem Content="3"/>
  <ListBoxItem Content="4"/>
  <ListBoxItem Content="5"/>
</ListBox>
```

Возможность выбора элементов, образующих *ListBox*, пользователем реализуется посредством события *SelectionChanged* (Выбранный элемент изменен) и свойства *SelectedItem* (Выбранный элемент). В предыдущем фрагменте XAML-кода обработкой события *SelectionChanged* занимался обработчик метода *ListBox_SelectionChanged*. Вот код:

```
private void ListBox_SelectionChanged(
    object sender, SelectionChangedEventArgs e)
{
  ListBoxItem x = theList.SelectedItem as ListBoxItem;
  string strTest = x.Content.ToString();
}
```

Обратите внимание, что свойство *SelectedItem* возвращает *object*, а не *ListBoxItem*, потому что (как упоминалось ранее) в *ListBox* могут размещаться разные типы содержимого. Поэтому, в данном случае, *SelectedItem* приводится к *ListBoxItem*.

Помните, что в *ListBox* может располагаться разное содержимое. Далее приведен пример, когда *ListBox* включает составные элементы, образованные компонентами *StackPanel*, каждый из которых содержит *Rectangle*, *Image* и *TextBlock*.

```
  <ListBox x:Name="theList" SelectionChanged="ListBox_SelectionChanged">
    <StackPanel Orientation="Horizontal">
      <Rectangle Fill="Black" Height="100" Width="100"></Rectangle>
```

```
        <Image Height="100" Width="100" Source="sl.jpg"/>
        <TextBlock Text="Item 1"></TextBlock>
      </StackPanel>
      <StackPanel Orientation="Horizontal">
        <Rectangle Fill="Black" Height="100" Width="100"></Rectangle>
        <Image Height="100" Width="100" Source="sl.jpg"/>
        <TextBlock Text="Item 2"></TextBlock>
      </StackPanel>
    </ListBox>
```

Как будет выглядеть этот *ListBox*, можно увидеть на рис. 8-1.

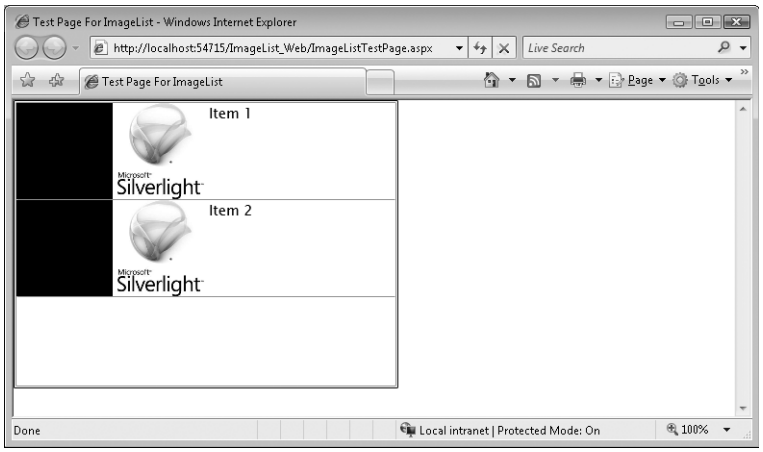

РИС. 8-1 *ListBox* со сложными элементами.

Содержимое таких сложных элементов, как представленные здесь, можно получить следующим образом: с помощью *SelectedItem* получить контейнер (в данном случае, *StackPanel*) и затем извлечь из него необходимое содержимое. Например:

```
StackPanel s = theList.SelectedItem as StackPanel;
TextBlock t = s.Children[2] as TextBlock;
string strTest = t.Text;
```

Элемент управления *RadioButton*

Элемент управления *RadioButton*, подобно *CheckBox* используется для фиксирования выбора пользователя. Однако отличается от *CheckBox* тем, что обычно применяется в ситуациях, когда пользователь должен выбрать *единственный* вариант из ряда предлагаемых опций.

Один из способов управления набором опций с возможностью единственного выбора – задать элементы управления *RadioButton*, соответствующие опциям, как элементы одного уровня в контейнере. Рассмотрим пример:

```
<StackPanel Orientation="Vertical" Background="Yellow">
  <RadioButton Content="Option 1" IsChecked="true" ></RadioButton>
  <RadioButton Content="Option 2"></RadioButton>
  <RadioButton Content="Option 3"></RadioButton>
  <RadioButton Content="Option 4"></RadioButton>
  <StackPanel Orientation="Vertical" Background="White">
    <RadioButton Content="Option 5" IsChecked="true"></RadioButton>
    <RadioButton Content="Option 6"></RadioButton>
    <RadioButton Content="Option 7"></RadioButton>
    <RadioButton Content="Option 8"></RadioButton>
  </StackPanel>
</StackPanel>
```

В данном XAML-коде имеется два контейнера *StackPanel*, один из которых расположен в другом. Первый контейнер содержит опции 1, 2, 3, и 4; второй – опции 5, 6, 7 и 8. В результате, пользователь может выбрать только одну из опций от 1 до 4 и только одну из опций от 5 до 8. Как это работает представлено на рис. 8-2.

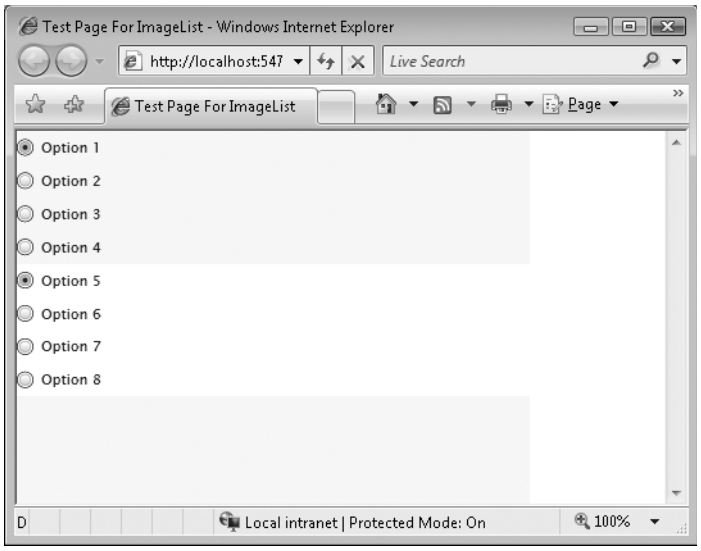

РИС. 8-2 Использование элемента управления *RadioButton*.

Кроме того, опции можно сгруппировать, задавая для каждого *RadioButton* свойство *GroupName* (Имя группы). В предыдущем примере в *StackPanel* располагались опции 1 -4, и Silverlight разрешал пользователю приложения выбирать только одну из них.

Если потребовалось бы разделить этот набор опций на две группы, чтобы пользователь мог выбирать одну из опций 1 и 2, и одну из опций 3 и 4, вместо контейнера можно было бы использовать группы *RadioButton*.

```
<StackPanel Orientation="Vertical" Background="Yellow">
  <RadioButton Content="Option 1" IsChecked="true"
     GroupName="G1" ></RadioButton>
  <RadioButton Content="Option 2" GroupName="G1" />
  <RadioButton Content="Option 3" GroupName="G2" />
  <RadioButton Content="Option 4" GroupName="G2" />
  <RadioButton Content="Option 5" GroupName="G3"
     IsChecked="true" />
  <RadioButton Content="Option 6" GroupName="G3" />
  <RadioButton Content="Option 7" GroupName="G3" />
  <RadioButton Content="Option 8" GroupName="G3" />
</StackPanel>
```

Как это выглядит в действии, можно увидеть на рис. 8-3.

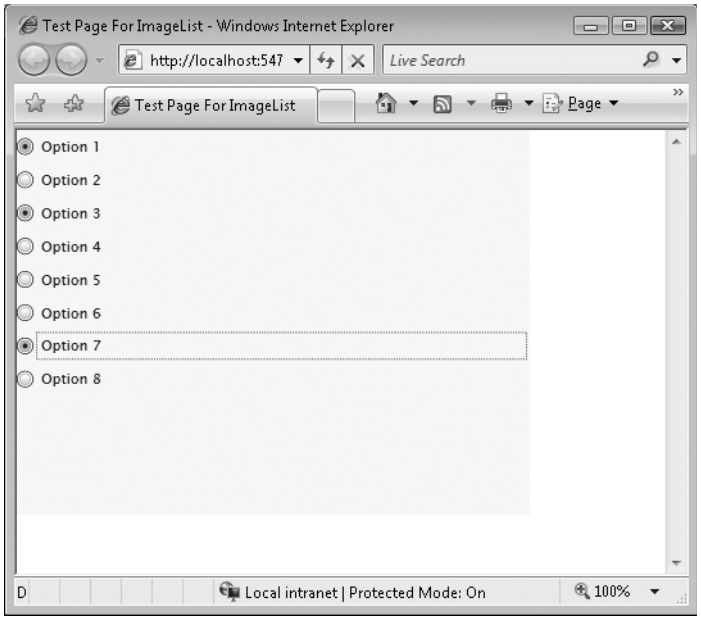

РИС. 8-3 Использование элемента управления *RadioButton* с группами.

Аналогично элементам управления *Button* и *CheckBox*, *RadioButton* имеет свойство *ClickMode*, определяющее, какое действие пользователя приводит к формированию события *Click*. Его возможными значениями являются: *Hover,* при котором событие формируется уже от того, что пользователь проводит курсором по элементу управления; *Press*, при котором событие формируется по щелчку кнопки мыши по элементу управления; и *Release*, при котором кнопка мыши должна быть нажата, а потом отпущена, при нахождении курсора мыши на кнопке выбора. Также аналогично элементу управления *CheckBox*, состояние выбора отдельного *RadioButton* может быть проверено с помощью свойства *IsChecked*.

Элемент управления *TextBlock*

Элемент управления *TextBlock* используется в приложениях Silverlight для отображения текста. В самом простом случае, для отображения текст достаточно применить *TextBlock* с заданным свойством *Text*. Например:

```
<TextBlock Text="1234"></TextBlock>
```

Размер текста можно менять, задавая свойство *FontSize* (в пикселах). Используемый шрифт определяется свойством *FontFamily*; например, XAML-код для вывода текста шрифтом Arial Black размером 20 выглядел бы так:

```
<TextBlock Text="1234" FontFamily="Arial Black" FontSize="20" />
```

Свойство *FontStyle* определяет, как будет выведен текст, курсивом или обычным шрифтом. Чтобы текст отображался курсивом, свойству *FontStyle* задается значение *Italic*:

```
<TextBlock Text="1234" FontFamily="Arial Black"
   FontSize="20" FontStyle="Italic"></TextBlock>
```

Чтобы использовать обычный шрифт, можно задать свойству *FontStyle* значение *Normal* или просто оставить его незаданным.

Разрывы строк и использование разных шрифтов для отображения текста реализуется с помощью подэлементов *TextBlock*: <*LineBreak*> и <*Run*>. Как следует из его имени, <*LineBreak*> создает разрывы в тексте с переносом остального текста на новую строку.

Однако *TextBlock* не является элементом управления для представления содержимого, как некоторые из элементов управления, которые были представлены в данной главе ранее. Поэтому если необходимо представить текст иначе, чем все остальное содержимое свойства *Text*, используется элемент управления *Run*:

```
<TextBlock Width="400" Text="My first text">
  <LineBreak/>
  <Run>My Second Text</Run>
  <LineBreak/>
  <Run>My Third Text</Run>
  <LineBreak/>
  <Run>My Fourth Text</Run>
</TextBlock>
```

Замечательное качество элемента управления *Run* в том, что он поддерживает те же свойства шрифта, размера, цвета и т.д., что и *TextBlock*. Таким образом, он предоставляет тот же уровень управления его содержимым, что и *TextBlock*. Это означает, что текст *TextBlock* и текст, отображаемый с помощью *Run*, полностью согласованы.

Silverlight руководствуется правилами замещения системных шрифтов, поэтому если шрифт представлен в системе, текст будет отображен этим шрифтом; если шрифт недоступен в системе, будет использоваться резервный шрифт, определенный операционной системой.

Рассмотрим пример некоторого XAML, созданного для отображения *TextBlock*, в котором содержится текст, представленный шрифтом Webdings, а также некоторый текст на китайском языке и иврите:

```
<StackPanel Orientation="Vertical" Background="Yellow">
   <TextBlock Width="400" Text="My first text">
     <LineBreak/>
     <Run FontFamily="Webdings">My Second Text</Run>
     <LineBreak/>
     <Run>微软助力基础教育</Run>
     <LineBreak/>
     <Run>דיון וקבוצות בישראלקהילות הפיתוח מרכז
     </Run>
   </TextBlock>
</StackPanel>
```

Как это выглядит, можно увидеть на рис. 8-4. Данная новая возможность абсолютно упростила разработку Silverlight-приложений с многоязычным пользовательским интерфейсом.

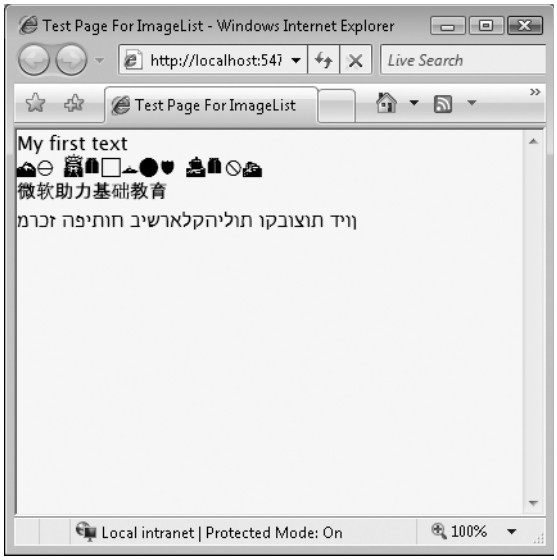

РИС. 8-4 Использование специальных шрифтов в Silverlight.

Элемент управления *TextBox*

Элемент управления *TextBox* обеспечивает пользователям область для ввода текста. В самой простой форме *TextBox* предоставляет область, в которую пользователь может ввести одну строку текста. Например:

```
<StackPanel>

  <TextBox />
  <TextBox />

</StackPanel>
```

Это очень простой *StackPanel*, содержащий два элемента управления *TextBox*. На рис. 8-5 показано, как это будет выглядеть. В текстовые поля введены некоторые значения для демонстрации того, как пользователь мог бы заполнять *TextBox* в приложении.

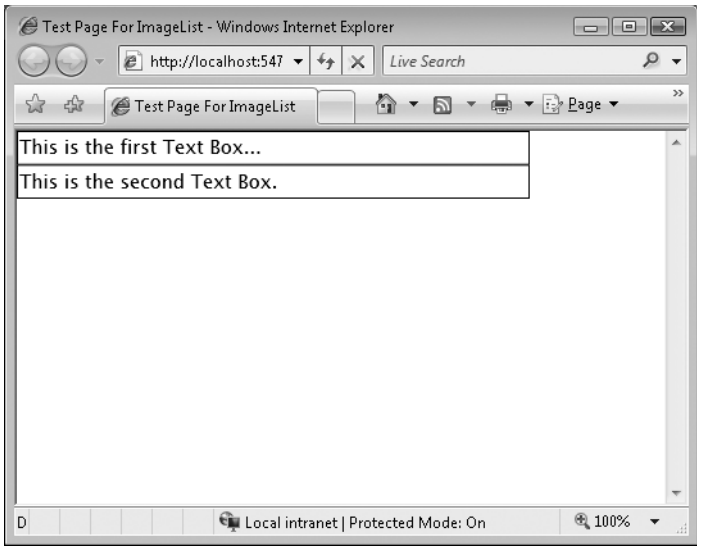

РИС. 8-5 Простые элементы управления *TextBox*.

Чтобы эти элементы управления *TextBox* могли принимать многостроковый ввод, понадобится сделать две вещи. Во-первых, необходимо задать высоту или поместить их в контейнер, обеспечивающий им нестандартную высоту. Затем свойству *AcceptsReturn* (Допускаются символы возврата каретки) потребуется задать значение *True*, благодаря чему эти элементы управления смогут принимать символы возврата каретки. Что будет получено в результате, можно увидеть на рис. 8-6.

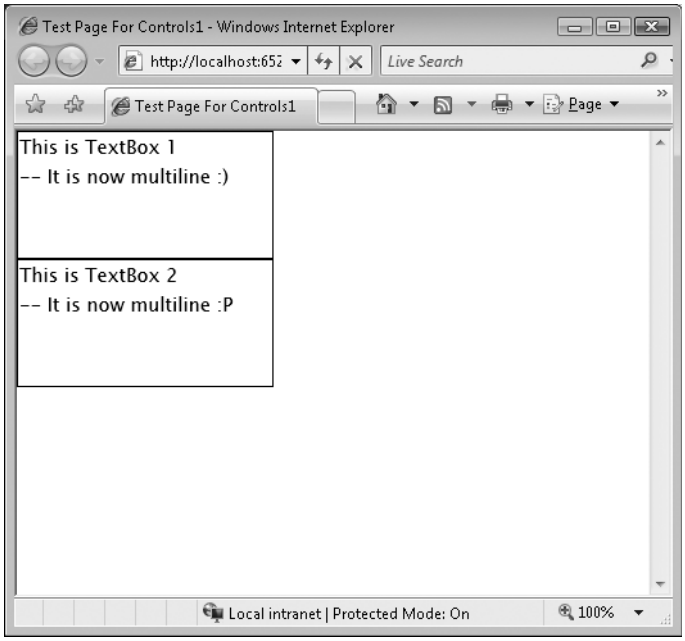

РИС. 8-6 Элементы управления *TextBox* настроенные для многострокового ввода текста пользователем.

Свойство *Text* такого *TextBox* возвращает объект типа *String*, в котором каждая строка будет отделена символом '\r'.

Каждый раз при изменении текста в *TextBox* формируется событие *TextChanged* (Текст изменен). Заметьте, это происходит после каждого нажатия клавиши, поэтому если необходимо отслеживать все изменения, как одно событие, возможно, лучше будет перехватывать событие *LostFocus* и работать с текстом из него.

Элементы управления *TextBox* автоматически принимают ввод через Редактор метода ввода (Input Method Editor, IME) для поддержки иностранных языков. Если в системе установлена поддержка различных клавиатур и языков , IME будет работать в Silverlight 2 точно так же, как для любого другого

элемента управления HTML. Это можно видеть на рис. 8-7, где в одном *TextBox* используются разные языки IME – изумительно!

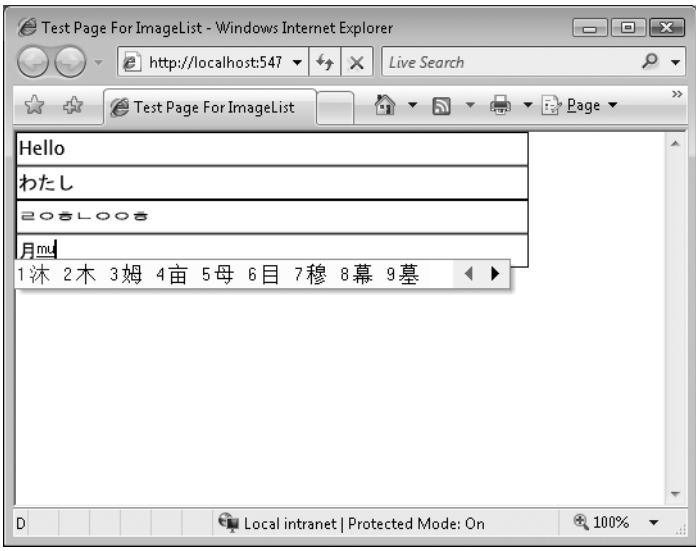

РИС. 8-7 Элемент управления *TextBox* с текстом на разных языках.

В данном случае, свойство *Text* возвращает строку с использованием кодировки Unicode со всеми символами и '\r' в качестве разделителей строк.

Элементы управления *TextBox* позволяют выбирать подстроки *Text* и также обеспечивают возможность настроить то, как выглядит выбранный текст. Для задания или возвращения выбранного в настоящий момент текста используется свойство *SelectedText* (Выбранный текст). Начало выбранной подстроки можно получить с помощью свойства *SelectionStart* (Начало выбранной подстроки), ее длину – с помощью свойства *SelectionEnd* (Конец выбранной подстроки). При любом изменении выбора подстроки формируется событие *SelectionChanged* (Выбранный элемент изменен). Рассмотрим следующий пример:

Во-первых, XAML-код:

```
<TextBox Height="100" AcceptsReturn="True"
  Text="ABCDEFGHIJKLMNOPQRSTUVWXYZ"
  SelectionChanged="TextBox_SelectionChanged">
</TextBox>
```

Здесь определяется событие *SelectionChanged*, для которого определен обработчик события *TextBox_SelectionChanged*. Рассмотрим:

```
private void TextBox_SelectionChanged(object sender, RoutedEventArgs e)
{
  TextBox t = sender as TextBox;
  int st = t.SelectionStart;
  int ln = t.SelectionLength;
  string strT = t.SelectedText;
}
```

В данном случае, переменная *st* типа *int* содержит индекс начального символа выбранной подстроки (отсчитываемой от нуля), *ln* – длина выбранной подстроки, и *strT* – строка, содержащая выбранную подстроку. Итак, например, если выбрана подстрока, как показано на рис. 8-8, *st* будет равна 9, *ln* равна 8, и *strT* будет равна JKLMNOPQ.

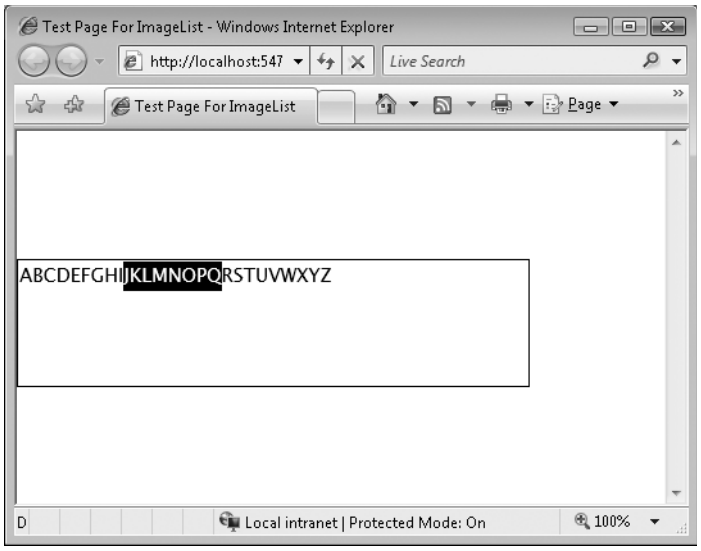

РИС. 8-8 Выбор подстроки текста.

Также в *TextBox* можно использовать разные шрифты, точно так же как это делается в *TextBlock*. Подробно об этом рассказывалось в разделе «Элемент управления *TextBlock*» в этой главе ранее.

Элемент управления *PasswordBox*

Очень похож на *TextBox* элемент управления *PasswordBox* (Поле пароля). *PasswordBox* во многих отношениях практически идентичен *TextBox*. С помощью свойства *PasswordChar* (Символ пароля) этого элемента управления можно задать символ, который будет использоваться вместо любого введенного символа для отображения нажатия клавиши.

Итак, например, *PasswordBox*, в котором вводимые символы заменяются символом # (знак числа), описывается следующим XAML-кодом:

```
<PasswordBox PasswordChar="#" x:Name="pw"></PasswordBox>
```

Кроме того, следует отметить, что вводимое пользователем содержимое сохраняется в свойстве *Password* (Пароль), а не в свойстве *Text*, поэтому для получения ввода пользователя должно использоваться свойство *Password*.

Общие свойства, события и методы

В предыдущих разделах были подробно рассмотрены базовые элементы управления Silverlight с акцентированием внимания на их особых свойствах, методах и событиях. Однако существует ряд функциональных возможностей, общих для всех или большинства этих элементов управления, и вы должны познакомиться с ними, прежде чем начнете разработку собственных приложений Silverlight.

Обработка фокуса

Элементы, которые могут принимать ввод, формируют события *GotFocus* и *LostFocus*. События *GotFocus* и *LostFocus* формируются каждый раз, когда пользователь переходит на или покидает элемент управления, либо наводя курсором мыши, либо используя клавишу Tab. Оба этих события являются событиями, *передающимися вверх по иерархии* (*bubbling* events). Это означает, что элемент управления принимает событие, но не обрабатывает его. Событие передается его родителю и так далее вверх по иерархии до тех пор, пока не будет перехвачено обработчиком события.

Обработка событий мыши

Элементы управления Silverlight формируют несколько событий мыши:

- ***MouseEnter*** Событие *MouseEnter* формируется при входе курсора мыши в область отображения элемента управления.
- ***MouseLeave*** Событие *MouseLeave* формируется, когда курсор мыши покидает область отображения элемента управления.
- ***MouseLeftButtonDown*** Событие *MouseLeftButtonDown* формируется при нажатии пользователем левой кнопки мыши, когда курсор находится над элементом управления.
- ***MouseLeftButtonUp*** Событие *MouseLeftButtonUp* формируется при отпускании удерживаемой левой кнопки мыши, когда курсор находится над элементом управления.
- ***MouseMove*** Событие *MouseMove* формируется при перемещении курсора по элементу управления.

Кроме перечисленных событий для обеспечения максимального контроля могут использоваться методы *CaptureMouse* и *ReleaseMouseCapture*. Когда для элемента управления вызывается метод *CaptureMouse*, все события мыши направляются в этот элемент управления независимо от того, находится ли курсор мыши в его границах или нет. Метод *ReleaseMouseCapture*, как следует из его имени, возвращает механизм перехвата событий в нормальное состояние. Эти методы очень полезны для реализации перемещения объектов методом «drag-and-drop». Далее представлен пример использования этих событий и методов.

Использование событий мыши для реализации перемещения объектов методом «Drag-and-drop»

Ниже представлен XAML-документ, обеспечивающий отрисовку эллипсов на странице. Обратите внимание, что для всех эллипсов определены одни и те же обработчики событий мыши:

```
<UserControl x:Class="MouseEvents.MainPage"
  xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
  FontFamily="Trebuchet MS" FontSize="11"
  Width="640" Height="480">
  <Grid x:Name="LayoutRoot" Background="White">
    <Canvas Width="640" Height="480" Background="Wheat">
      <Ellipse Canvas.Top="0" Fill="Black" Width="20" Height="20"
        MouseLeftButtonDown="Ellipse_MouseLeftButtonDown"
        MouseLeftButtonUp="Ellipse_MouseLeftButtonUp"
        MouseMove="Ellipse_MouseMove" />
      <Ellipse Canvas.Top="40" Fill="Black" Width="20" Height="20"
        MouseLeftButtonDown="Ellipse_MouseLeftButtonDown"
        MouseLeftButtonUp="Ellipse_MouseLeftButtonUp"
        MouseMove="Ellipse_MouseMove" />
      <Ellipse Canvas.Top="80" Fill="Black" Width="20" Height="20"
        MouseLeftButtonDown="Ellipse_MouseLeftButtonDown"
        MouseLeftButtonUp="Ellipse_MouseLeftButtonUp"
        MouseMove="Ellipse_MouseMove" />
      <Ellipse Canvas.Top="120" Fill="Black" Width="20" Height="20"
        MouseLeftButtonDown="Ellipse_MouseLeftButtonDown"
        MouseLeftButtonUp="Ellipse_MouseLeftButtonUp"
        MouseMove="Ellipse_MouseMove" />
    </Canvas>
  </Grid>
</UserControl>
```

Теперь рассмотрим выделенный код. Во-первых, нам необходимы переменные для сохранения x и y координат курсора мыши в момент начала перемещения методом «drag-and-drop» и еще одна переменная для обозначения того, удерживается ли кнопка мыши в настоящий момент.

```
double beginX = 0;
double beginY = 0;
bool isMouseDown = false;
```

Если курсор находится над любой из кнопок, и нажимается кнопка мыши, выполняется функция *Ellipse_MouseLeftButtonDown*.

```
private void Ellipse_MouseLeftButtonDown(object sender,
      MouseButtonEventArgs e)
{
   Ellipse b = sender as Ellipse;
   beginX = e.GetPosition(this).X;
   beginY = e.GetPosition(this).Y;
   isMouseDown = true;
   b.CaptureMouse();
}
```

Отправителем является универсальный объект, поэтому, прежде всего, необходимо привести его к типу эллипс. После этого используем аргумент *MouseButtonEventArgs* (Аргументы событий кнопки мыши) для получения текущих координат x и y и задания их как начальных значений. Затем, поскольку кнопка мыши удерживается нажатой, ее перемещения должны отслеживаться, поэтому переменной *isMouseDown* задается значение *True*. Наконец, мы хотим отслеживать события мыши для этого эллипса (любого из заданных) таким образом, чтобы даже если пользователь уводит курсор мыши с этого эллипса, он продолжал получать события.

Итак, поскольку пользователь двигает мышью, должно формироваться событие *MouseMove*. Оно обрабатывается функцией-обработчиком события *Ellipse_MouseMove*:

```
private void Ellipse_MouseMove(object sender,
      MouseEventArgs e)
{
   if (isMouseDown)
   {
      Ellipse b = sender as Ellipse;
      double currX = e.GetPosition(this).X;
      double currY = e.GetPosition(this).Y;
      b.SetValue(Canvas.LeftProperty, currX);
      b.SetValue(Canvas.TopProperty, currY);
   }
}
```

Это событие формируется при прохождении курсора мыши по эллипсу, независимо от того, выполняется ли перемещение этого эллипса или нет. Но нас интересуют только варианты, когда пользователи перетягивает эллипс, а это, по определению, означает, что переменная *isMouseDown* имеет значение *True*. В этом случае мы получаем текущие координаты курсора мыши и используем их для задания присоединенных свойств *Top* и *Left* эллипса. В результате будет создан эффект перетягивания эллипса соответственно координатам курсора мыши.

Наконец, при отпускании удерживаемой кнопки мыши формируется событие *MouseLeftButtonUp*, перехватываемое функцией *Ellipse_MouseLeftButtonUp*:

```
private void Ellipse_MouseLeftButtonUp(object sender,
      MouseButtonEventArgs e)
{
   Ellipse b = sender as Ellipse;
   isMouseDown = false;
   b.ReleaseMouseCapture();
}
```

Это обеспечивает возвращение переменной *isMouseDown* к исходному значению, и элемент управления, отслеживающий события мыши, прекращает делать это. В результате восстанавливается обычное поведение мыши, а эллипс остается передвинутым в другое место. Все это в действии можно увидеть на рис. 8-9 и 8-10.

На рис. 8-9 показано, как приложение выглядит в исходном состоянии. Если пользователь наведет курсор мыши на любой из эллипсов и затем нажмет кнопку мыши и продолжит перемещение, эллипс начнет двигаться за курсором. Таким образом, пользователь может менять расположение эллипсов на экране методом «drag-and-drop». Это представлено на рис. 8-10

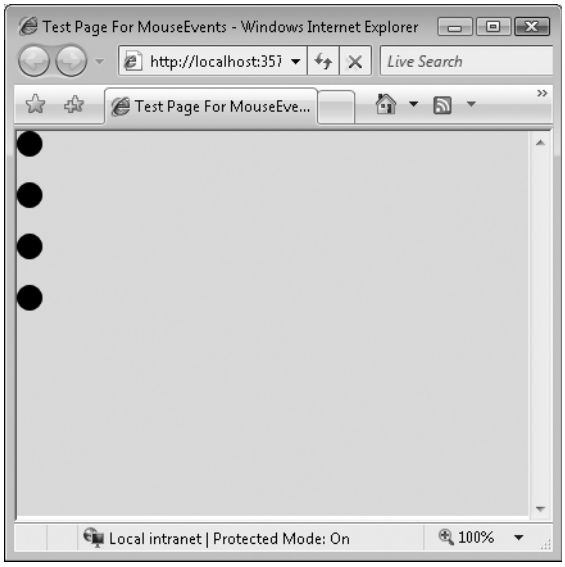

РИС. 8-9 Приложение, реализующее перемещение объектов методом «Drag-and-drop».

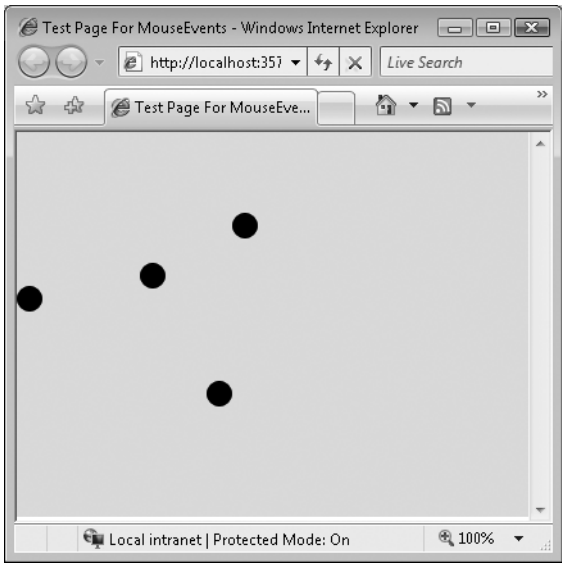

РИС. 8-10 Перемещение элементов управления методом «drag-and-drop».

Кроме событий мыши, существуют события клавиатуры, которые могут формироваться для многих элементов управления. Подробнее об этом рассказывается в следующем разделе.

Использование клавиатуры

Кроме элемента управления *TextBox*, который предлагает пользователю возможность ввода текста с клавиатуры, многие элементы управления формируют события *KeyDown* и *KeyUp*, которые могут использоваться для перехвата ввода с клавиатуры. Событие *KeyDown* формируется при каждом нажатии клавиши, когда фокус находится на элементе. Функция-обработчик этого события принимает универсальный объект-отправитель и объект *KeyEventArgs*, который может использоваться для извлечения информации о нажатой клавише. Он имеет свойство *key*, возвращающее объект *Key* с информацией о клавише. Это платформонезависимое свойство, поэтому оно лучше всего подходит для использования в приложениях Silverlight. *KeyEventArgs* предлагает также еще одно свойство,

PlatformKeyCode (Код клавиши на данной платформе), которое может использоваться со свойством *key* для получения кодов клавиш, используемых в конкретной операционной системе.

Кроме того, статическое свойство *Keyboard.Modifers* возвращает значение *ModifierKeys* (Модифицирующие клавиши), которое может использоваться для определения того, какая из модифицирующих клавиш (Alt, Ctrl, Shift, Windows или Apple) была нажата. Просто используйте побитовое И на возвращенное значение свойства для проверки искомой модифицирующей клавиши (например, Shift).

Заключение

В данной главе представлены некоторые основные элементы управления Silverlight (*Button, CheckBox, ComboBox, HyperlinkButton*, *Image, ListBox, RadioButton, TextBlock*, *TextBox* и *PasswordBox*) и показано, как они могут использоваться для создания приложений Silverlight. Кроме того, дан обзор свойств, методов и событий, общих для всех элементов управления, что поможет вам начать использовать элементы управления представления и компоновочные элементы управления в Silverlight.

В следующей главе рассматриваются расширенные элементы управления Silverlight и рассказывается, как они могут использоваться для расширения круга возможностей ваших приложений.

Глава 9

Дополнительные элементы управления Silverlight

В Главе 8, «Основные элементы управления Silverlight», представлен набор элементов управления Microsoft Silverlight и проведен обзор таких основных элементов управления, как *TextBox* и *CheckBox*. Однако для современных насыщенных интерактивных приложений (RIAs) необходимы намного более сложные элементы управления, такие как таблицы с данными, календари и т.д. В данной главе вы научитесь использовать эти элементы управления в своих Silverlight приложениях. Некоторые из этих элементов управления, такие как *DataGrid* (Таблица с данными), заслуживают отдельной книги, поэтому не ожидайте получить здесь полный API этих элементов управления. Однако в данной главе предоставлено достаточно сведений об этих расширенных элементах управления, которые позволят понимать, что происходит при их использовании. Располагая этими знаниями, можно продолжать экспериментировать и изучать эти элементы управления уже самостоятельно.

Элемент управления *DataGrid*

Элемент управления *DataGrid* разработан для облегчения задачи по отображению данных в формате таблицы. Это сетка *данных*, а не просто сетка, потому что она может быть связана с источником данных. Более подробно отображение данных рассматривается в Главе 12, «Создание взаимодействующих с сервером приложений в Silverlight», но чтобы вы могли осознать мощь *DataGrid*, некоторые из методик работы с ним будут использованы здесь.

При добавлении элемента управления *DataGrid*[1] в окно редактирования Extensible Application Markup Language (XAML), получаем такой XAML-код:

```
<UserControl
  xmlns:data="clr-namespace:System.Windows.Controls;assembly=System.Windows.Controls.Data"
        x:Class="SilverlightApplication1.MainPage"
  xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
  Width="640" Height="480">
  <Grid x:Name="LayoutRoot" Background="White">
    <data:DataGrid></data:DataGrid>
  </Grid>
</UserControl>
```

Пока что *DataGrid* не может делать практически ничего, поэтому добавим в него два свойства: имя и *AutoGenerateColumns* (Столбцы формируются автоматически), которое обеспечит автоматическое создание необходимых столбцов данных при привязке сетки к *DataGrid*. Вот как выглядит XAML для *DataGrid* после этих дополнений:

```
<data:DataGrid x:Name="GrdHeadline" AutoGenerateColumns="True">
</data:DataGrid>
```

Вот и все, что требуется для подготовки *DataGrid* к использованию связанных данных. Прежде чем переходить к остальным свойствам, методам и событиям этого элемента управления, давайте выполним простую привязку данных. Будем использовать RSS-канал по адресу *http://feeds.reuters.com/reuters/oddlyEnoughNews?format=xml*.

---

[1] Необходимо добавить ссылку на System.Windows.Controls.Data (*прим. редактора*).

Далее добавим на страницу кнопку и настроим обработчик события *Click*. Также нам потребуется настроить объект *WebClient* (Веб-клиент) на чтение данных по этому URI и добавить обработчик события для обработки завершения загрузки данных. Для этого добавляем следующий код:

```
WebClient client = new WebClient();
Uri uri =
  new Uri("http://feeds.reuters.com/reuters/topNews?format=xml");
client.DownloadStringCompleted +=
  new DownloadStringCompletedEventHandler(
      client_DownloadStringCompleted);
client.DownloadStringAsync(uri);
```

Теперь по завершении загрузки данных будет запускаться обработчик события *client_DownloadStringCompleted*. В этой функции будет использоваться специальный класс типа *NewsHeadline* (Заголовок новостей). Вот код этого класса:

```
public class NewsHeadLine
{
 public string strHead { get; set; }
 public string strLine { get; set; }
}
```

client_DownloadStringCompleted использует библиотеки System.Xml.Linq, поэтому понадобится добавить ссылку на них. Для этого щелкните правой кнопкой мыши References и выберите Add Reference. На вкладке .NET вы увидите библиотеку System.Xml.Linq, как показано на рис. 9-1. Выберите ее и щелкните OK.

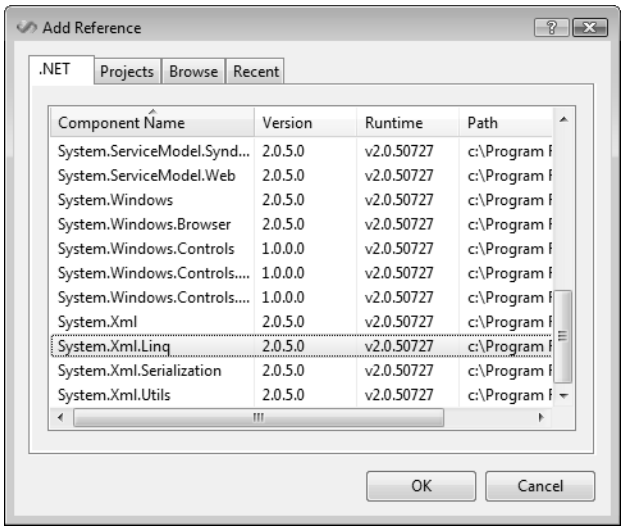

РИС. 9-1 Добавление ссылки на System.Xml.Linq.

Чтобы код скомпилировался, в нем должна быть указана эта библиотека. Для этого просто добавляем данное выражение в начало MainPage.xaml.cs:

```
using System.Xml.Linq;
```

Итак, получаем код обработчика событий *client_DownloadStringCompleted*:

```
void client_DownloadStringCompleted(
 object sender, DownloadStringCompletedEventArgs e)
{
 XDocument xmlHeadlines = XDocument.Parse(e.Result);
 var headlines = from story in xmlHeadlines.Descendants("item")
              select new NewsHeadLine
              {
                strHead = (string)story.Element("title"),
                strLine = (string)story.Element("link")
              };
            Deployment.Current.Dispatcher.BeginInvoke(() =>
             {
 GrdHeadline.ItemsSource = headlines;
```

```
        });
}
```

Этот код обеспечивает синтаксический разбор данных, возвращаемых в результате запроса к сервису (хранящихся в e.Result), в объект *XDocument*. После этого с помощью LINQ данные выбираются из *XDocument* и создается коллекция заголовков. Затем эта коллекция присваивается как значение свойству *ItemsSource* (Источник элементов) элемента управления *DataGrid*. Результаты представлены на рис. 9-2.

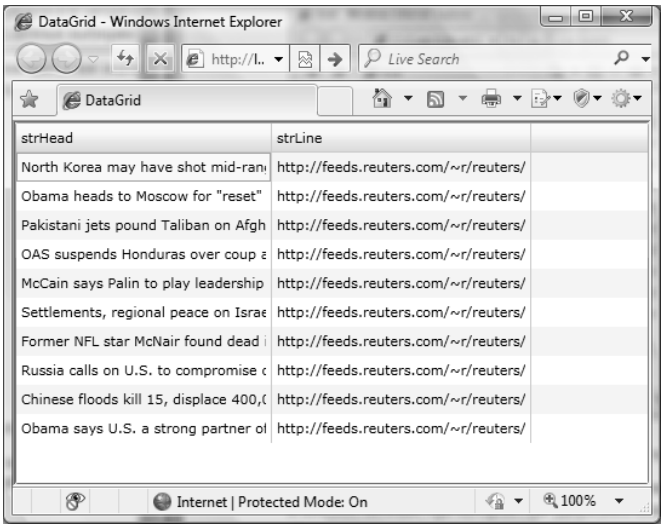

РИС. 9-2 Образец коллекции заголовков, созданный при выполнении элемента управления *DataGrid*.

В *DataGrid* возможны два режима выбора: одиночный, при котором можно выбирать по одной строке, и множественный, при котором при удержании клавиш Ctrl и/или Shift может быть выбрано несколько строк.

Эти режимы задаются с помощью свойства *SelectionMode* (Режим выбора), которое принимает значения *DataGridSelectionMode.Single* для одиночного выбора и *DataGridSelectionMode.Extended* для множественного.

При использовании единичного режима выбранный элемент сохраняется в *SelectedItem*. Если этот объект типа *object*, перед использованием он должен быть приведен к соответствующему типу. Вы видели, что сетка заполнялась элементами определенного нами класса *NewsHeadLine*.

Итак, для единичного выбора выбранные данные могут быть получены следующим образом:

```
NewsHeadLine theHeadline = GrdHeadline.SelectedItem as NewsHeadLine;
```

Когда установлен режим множественного выбора, можно использовать свойство *SelectedItems*, возвращающее коллекцию списков. Это очень просто сделать, вот код:

```
string strHead;
string strLink;
System.Collections.IList listOfItems = GrdHeadline.SelectedItems;
foreach(NewsHeadLine newsHead in listOfItems)
{
  strHead = newsHead.strHead;
  strLink = newsHead.strLine;
}
```

В данном случае, свойство *SelectedItems* возвращает *System.Collections.IList*. После этого можно перебрать элементы этого списка, извлекая каждый из объектов *NewsHeadline* и получая его данные.

При использовании любой таблицы, и *DataGrid* здесь не исключение, всегда полезно сделать ее более удобной для восприятия путем использования цветных полос, т.е. чередовать закрашенную и незакрашенную строки попеременно разными цветами или оттенками. В данном случае, чтобы

сделать *DataGrid* полосатым, можно использовать свойства *RowBackground* (Фон строки) и *AlternatingRowBackground* (Чередующийся фон строки). В качестве значений этим свойствам задается цвет кисти следующим образом:

```
GrdHeadline.RowBackground =
   new SolidColorBrush(Colors.LightGray);
GrdHeadline.AlternatingRowBackground =
new SolidColorBrush(Colors.Yellow);
```

Результат такого форматирования можно увидеть на рис. 9-3.

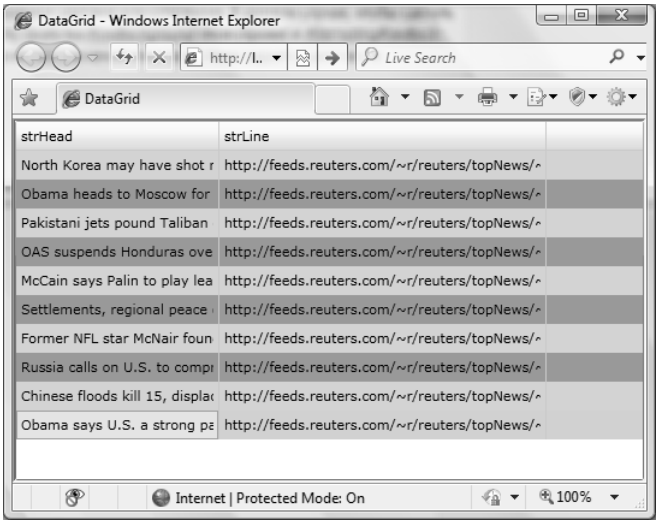

РИС. 9-3 Чередование цветов строк.

Возможно, вы обратили внимание, что столбцы добавляются в том порядке, в котором данные определены в классе *NewsHeadLine*. Однако порядок вывода данных на экран можно менять. Настройку по умолчанию можно переопределить свойством *DisplayIndex* (Индекс отображения). Например, имеется *DataGrid* с двумя столбцами. В первом выводятся заголовки, во втором – ссылки. Их можно поменять местами следующим способом:

```
GrdHeadline.Columns[0].DisplayIndex = 1;
GrdHeadline.Columns[1].DisplayIndex = 0;
```

Более широкие возможности управления визуальным представлением данных в сетке предоставляет шаблон данных. Шаблон данных – это XAML, определяющий то, как должны быть организованы данные, включая привязку к конкретным полям. Например, в предыдущих примерах вывода каждая ячейка таблицы была связана с определенным элементом RSS-канала. Чтобы вывести два поля RSS-канала в одной ячейке, необходимо использовать шаблон данных. Рассмотрим следующий XAML:

```
<UserControl
xmlns:data="clr-namespace:System.Windows.Controls;assembly=System.Windows.Controls.Data"
x:Class="DataGrid.MainPage"
  xmlns="http://schemas.microsoft.com/client/2007"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
  Width="640" Height="480">

  <Grid x:Name="LayoutRoot" Background="White">
    <data:DataGrid x:Name="GrdHeadline" AutoGenerateColumns="True">
      <data:DataGrid.Columns>
        <data:DataGridTemplateColumn>
          <data:DataGridTemplateColumn.CellTemplate>
            <DataTemplate>
              <StackPanel Orientation="Vertical">
              <TextBlock Text="123"></TextBlock>
              <TextBlock Text="{Binding strHead}"></TextBlock>
              <TextBlock Text="{Binding strPubDate}"></TextBlock>
              </StackPanel>
            </DataTemplate>
          </data:DataGridTemplateColumn.CellTemplate>
```

```
        </data:DataGridTemplateColumn>
        <data:DataGridTemplateColumn>
          <data:DataGridTemplateColumn.CellTemplate>
            <DataTemplate>
              <TextBlock Text="{Binding strLine}"></TextBlock>
            </DataTemplate>
          </data:DataGridTemplateColumn.CellTemplate>
        </data:DataGridTemplateColumn>
      </data:DataGrid.Columns>
    </data:DataGrid>
  </Grid>
</UserControl>
```

В этом XAML столбцы *DataGrid* заданы заранее с помощью элемента <*data:DataGrid.Columns*>. Мы можем переопределить шаблон для столбцов по умолчанию, когда в ячейке выводится одно поле, задавая для каждой ячейки <*data:DataGridTemplateColumn.CellTemplate*>, в котором имеется код, описывающий, как должна отображаться ячейка.

Как видите, описано только две ячейки, поэтому данная таблица, независимо от количества полей данных, будет иметь только два столбца. Первый столбец, определенный шаблоном первой ячейки, будет представлен элементом *StackPanel*, содержащим три текстовых поля, в которых выводятся заданное в коде «123», связанное с данными свойство *strHead* класса *NewsHeadline* и связанное с данными свойство *strPubDate* того же класса *NewsHeadline*. *StackPanel* выводит эти поля друг над другом.

Второй столбец, описанный шаблоном второй ячейки, является простым *TextBlock*, связанным с полем *strLine*. Возможно, вы заметили, что здесь появилось еще одно поле, *strPubDate*, которого не было в исходном примере класса, поэтому описание класса и код привязки необходимо обновить. Вот новое описание класса:

```
public class NewsHeadLine
   {
      public string strHead { get; set; }
      public string strLine { get; set; }
      public string strDescription { get; set; }
      public string strPubDate { get; set; }
   }
```

И вот новый код привязки данных, который обеспечивает привязку дополнительных полей:

```
void client_DownloadStringCompleted(object sender, DownloadStringCompletedEventArgs e)
{
  XDocument xmlHeadlines = XDocument.Parse(e.Result);
  var headlines = from story in xmlHeadlines.Descendants("item")
                  select new NewsHeadLine
                  {
                    strHead = (string)story.Element("title"),
                    strLine = (string)story.Element("link"),
                    strDescription = (string)story.Element("description"),
                    strPubDate = (string)story.Element("pubDate")
                  };
              Deployment.Current.Dispatcher.BeginInvoke(() =>
                {
  GrdHeadline.ItemsSource = headlines;
              });
}
```

Чтобы увидеть новые данные, необходимо внести еще одну небольшую коррективу – изменить высоту строки. Сейчас в ней может отображаться только одна строка текста, в результате чего содержимое первой ячейки будет обрезано по вертикали. Также понадобиться отключить автоматическое определение столбцов, потому что мы хотим переопределить поведение привязки данных *DataGrid* по умолчанию. Это обеспечивает следующий код (поместите его в конструктор *Page*):

```
GrdHeadline.AutoGenerateColumns = false;
GrdHeadline.RowHeight = 60;
```

Теперь, после выполнения этого кода, на экран будет выведен *DataGrid* с ячейками, соответствующими заданным шаблонам (рис. 9-4). Выглядит не очень красиво, но замечательно иллюстрирует возможности управления представлением данных в элементе управления *DataGrid*.

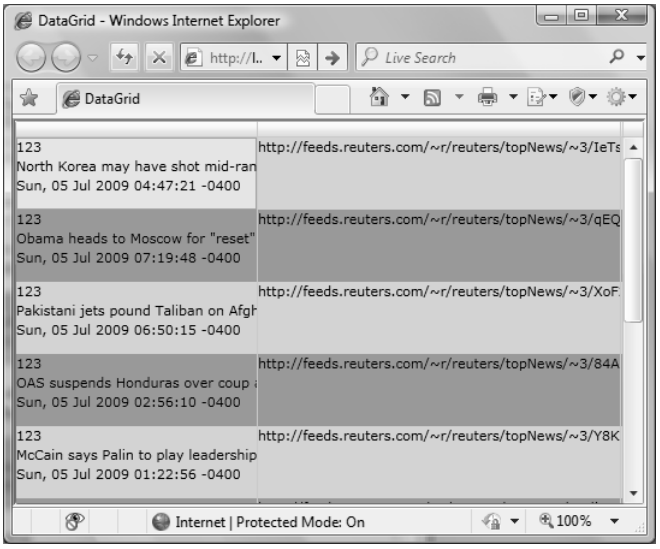

РИС. 9-4 Использование шаблона данных для определения визуального представления данных.

Элемент управления *DataGrid* обладает богатейшим API, который заслуживает отдельной книги. Однако того, что было рассказано о нем на этих нескольких страницах, достаточно, чтобы понять, какую невероятную гибкость он обеспечивает. Вы можете использовать эту базовую информацию для перехода на следующий уровень при создании собственных приложений Silverlight!

Элементы управления *Calendar* и *DatePicker*

Возможность использования дат в приложениях Silverlight обеспечивается элементами управления *Calendar* (Календарь) и *DatePicker* (Элемент выбора дат). Элемент управления *Calendar* выводит на экран дни заданного месяца или месяцы заданного года и предоставляет кнопки со стрелками для перемещения по месяцам или годам. *DatePicker* сочетает это представление с текстовым полем, в которое пользователь может вводить дату в соответствующем формате, или выпадающим списком, в котором можно выбрать определенную дату.

Рассмотрим сначала элемент управления *Calendar*. Он прост, поэтому удобно начинать с него:

```
<UserControl
  xmlns:controls="clr-namespace:System.Windows.Controls;assembly=System.Windows.Controls"
  x:Class="SilverlightApplication2.MainPage"
  xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
  Width="640" Height="480">
  <Grid x:Name="LayoutRoot" Background="White">
     <controls:Calendar x:Name="cal"></controls:Calendar>
  </Grid>
</UserControl>
```

Это обеспечит отображение *Calendar* в стандартном представлении месяца с выделенным соответственно текущему числу днем. Пример можно увидеть на рис. 9-5.

РИС. 9-5 Использование элемента управления *Calendar*.

Месяц отображаемой даты задается свойством *DisplayDate* (Отображаемая дата). Например, следующий код обеспечивает выбор даты 1 января 2009 года:

```
DateTime d = new DateTime(2009, 1, 1);
cal.DisplayDate = d;
```

Это обеспечит отображение выбранного месяца, но без указания конкретного дня, как показано на рис. 9-6.

Сравните рис. 9-6 и рис. 9-5. Можно заметить, что на рис. 9-6 дата не выбрана. При задании отображаемой даты выбранная дата была обнулена, поэтому свойство *SelectedDate* (Выбранная дата) имеет значение *null*. Если вы выберете какой-то день на экране, свойству *SelectedDate* будет присвоено это значение (типа *DateTime*), или дату можно задать в коде через все то же свойство *SelectedDate*. Например:

```
DateTime s = new DateTime(2009, 1, 31);
cal.SelectedDate = s;
```

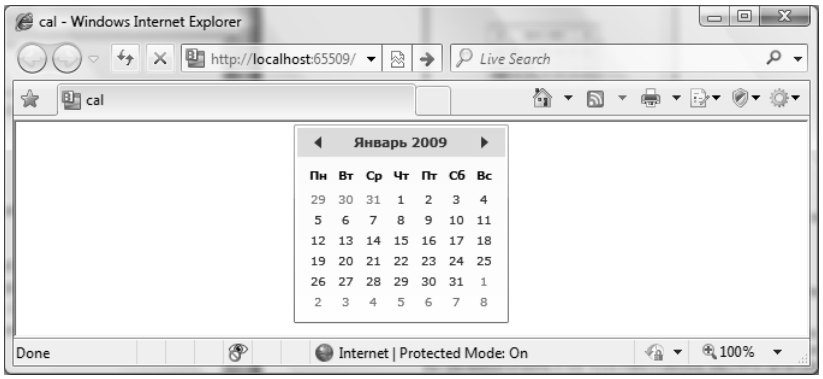

РИС. 9-6 Изменение отображаемой даты.

Как это выглядит, можно увидеть на рис. 9-7. Теперь на отображаемом календаре выбрана дата, 31 января.

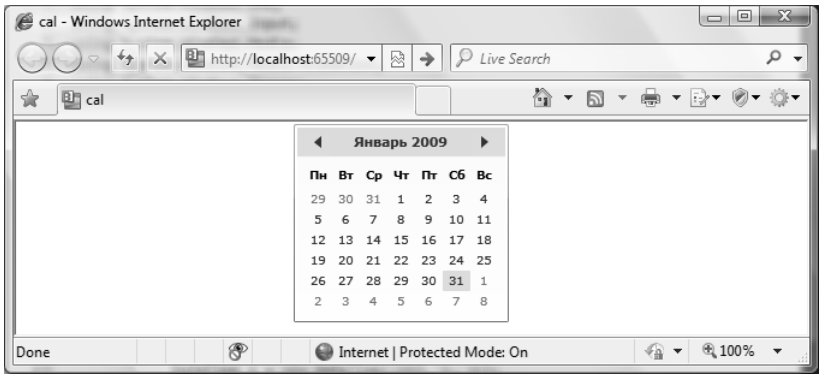

РИС. 9-7 Использование свойства *SelectedDate*.

Коллекция *BlackoutDates* (Недоступные даты) позволяет задавать диапазон дат, которые не могут быть выбраны. Эта коллекция содержит тип *CalendarDateRangeCollection* (Коллекция диапазона дат календаря), в который могут добавляться диапазоны дат. Все даты, попадающие в этот диапазон, будут отображаться неактивными и, таким образом, не могут быть выбраны. Свойства *SelectableDateStart* (Начало дат, доступных для выбора) и *SelectableDateEnd* (Конец дат, доступных для выбора) задают диапазон отображаемых дат. Например, с помощью этих свойств можно обеспечить вывод на экран календарь, содержащий диапазон отображаемых дат и диапазон (обычно поддиапазон) дат, доступных для выбора.

Элемент управления *DatePicker* сочетает функциональность календаря с текстовым полем. Свойство *Text* этого поля содержит все, что пользователь вводит в качестве даты. Если невозможно выполнить синтаксический разбор введенных пользователем данных, формируется ошибка *TextParseError* (Ошибка синтаксического разбора текста).

Рассмотрим XAML-код для *DatePicker*:

```
<Grid x:Name="LayoutRoot" Background="White">
   <controls:DatePicker x:Name="dp"></controls:DatePicker>
</Grid>
```

Визуальное представление данного фрагмента кода включает текстовое поле с датой, и справа от него располагается значок, выбор которого обеспечивает отображение элемента управления *Calendar* для выбора даты. Когда пользователь щелкает этот значок, формируется событие *CalendarOpened* (Календарь открыт). Когда пользователь закрывает его, формируется событие *CalendarClosed* (Календарь закрыт). Элементы управления *Calendar* и *DatePicker* обеспечивают базовые средства для создания приложений, в которых ввод основывается на датах и которые могут отображать информацию календаря.

Элементы управления *RepeatButton* и *ToggleButton*

Элемент управления *RepeatButton* (Кнопка повтора) предоставляет кнопку, при нажатии которой формируется множество событий щелчка. Частота формирования событий определяется свойством *Delay* (Задержка), оно задается в миллисекундах. Например, рассмотрим следующий XAML:

```
<RepeatButton x:Name="rpt" Delay="100"
  Content="0" Click="rpt_Click">
</RepeatButton>
```

В этом фрагменте задается код кнопки повтора *rpt* с задержкой 100 миллисекунд, содержимое которой по умолчанию равно 0. Обработка щелчка пользователя находится в обработчике событий *rpt_Click*. Вот код этого обработчика событий:

```
private void rpt_Click(object sender, RoutedEventArgs e)
{
  int n = Convert.ToInt16(rpt.Content);
  n++;
  rpt.Content = n.ToString();
}
```

Когда пользователь щелкает кнопку, текущее значение свойства *Content* извлекается, преобразовывается в целое число, увеличивается на единицу и опять присваивается как значение свойства *Content*. В результате надпись на кнопке увеличивается, пока пользователь удерживает кнопку мыши нажатой над *RepeatButton*. Это может использоваться как способ контроля при обработке множества повторяющихся действий. Например, для различных типов счетчиков, реализованных таким образом, что пользователю предоставляется кнопка со стрелкой, и, пока он удерживает ее, значение увеличивается.

Во всех остальных отношениях, *RepeatButton* является обычной кнопкой, поэтому все детали о его использовании можно найти в Главе 8, в которой рассматривается универсальный элемент управления *Button*.

ToggleButton (Выключатель) – это кнопка, которая, будучи нажатой, остается в этом состоянии до тех пор, пока по ней не щелкнут снова. Этот элемент управления сочетает в себе переключатель и обычную кнопку. Свойства элемента управления *ToggleButton* подобны используемым в переключателях и обычных кнопках: *ToggleButton* тоже имеет свойство *IsChecked*, с помощью которого можно извлекать его текущее значение, и свойство *IsThreeState*, которое посредством значения *Null* позволяет задавать третье состояние, среднее между обычным и нажатым.

Как и стандартный элемент управления *Button*, *RepeatButton* и *ToggleButton* являются элементами управления с содержимым, т.е. могут использоваться для отображения сложного содержимого и создания кнопок с изображениями, видеокнопок и многого другого для ваших приложений Silverlight.

Элемент управления *ScrollViewer*

Элемент управления *ScrollViewer* является контейнером для отображения информации. Он предоставляет горизонтальную и вертикальную полосы прокрутки, что обеспечивает возможность пользователю перемещать область отображения и просматривать все содержимое, если его размеры превышают размеры контейнера. Наличие полос прокрутки определяется свойствами *HorizontalScrollBarVisibility* (Видимость горизонтальной полосы прокрутки) и *VerticalScrollBarVisibility* (Видимость вертикальной полосы прокрутки). По умолчанию, если содержимое превышает размеры *ScrollViewer*, отображается вертикальная полоса прокрутки; горизонтальная полоса прокрутки не отображается независимо от содержимого. Рассмотрим следующий пример XAML:

```
<UserControl x:Class="sviewer.MainPage"
   xmlns="http://schemas.microsoft.com/client/2007"
   xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
   Width="640" Height="480">
   <Grid x:Name="LayoutRoot" Background="White">
      <ScrollViewer>
         <Image Source="mix08_1280.jpg" Stretch="None" />
      </ScrollViewer>
   </Grid>
</UserControl>
```

В данном случае *ScrollViewer* является единственным дочерним элементом управления на рабочей поверхности размером 640 × 480, поэтому его размеры также будут 640 × 480. В нем располагается изображение размером 1280 × 1024. Результат можно увидеть на рис. 9-8.

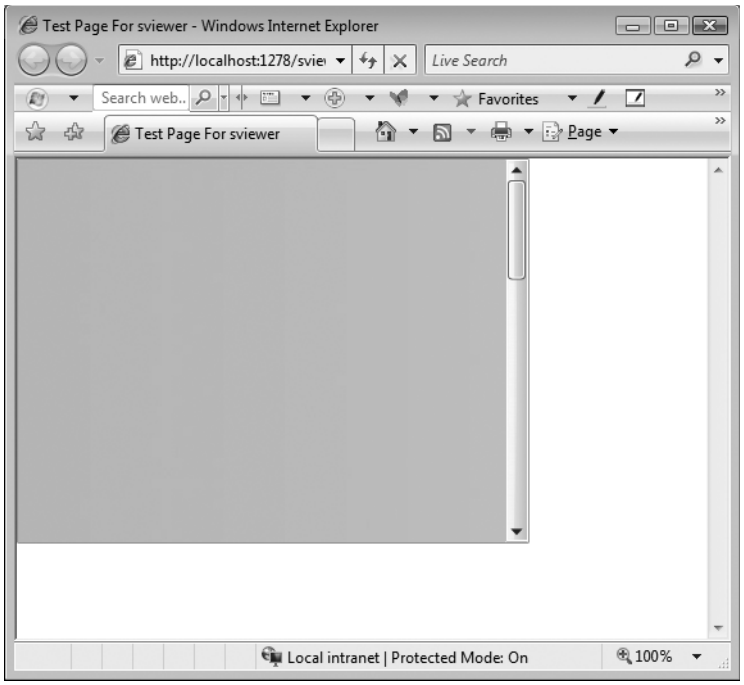

РИС. 9-8 Использование элемента управления *ScrollViewer*.

Как видно на рис. 9-8, отображается только закрашенная серым область, потому что видна только левая часть изображения. Горизонтальная полоса прокрутки отсутствует, так что мы не можем переместиться вправо к цветной части рисунка.

Но если в описание *ScrollViewer* в XAML добавить *HorizontalScrollBarVisibility="True"*, пользователь сможет увидеть все остальное изображение. Вот этот XAML-код:

```
<ScrollViewer HorizontalScrollBarVisibility="Visible">
  <Image Source="mix08_1280.jpg" Stretch="None" />
</ScrollViewer>
```

Результаты представлены на рис. 9-9.

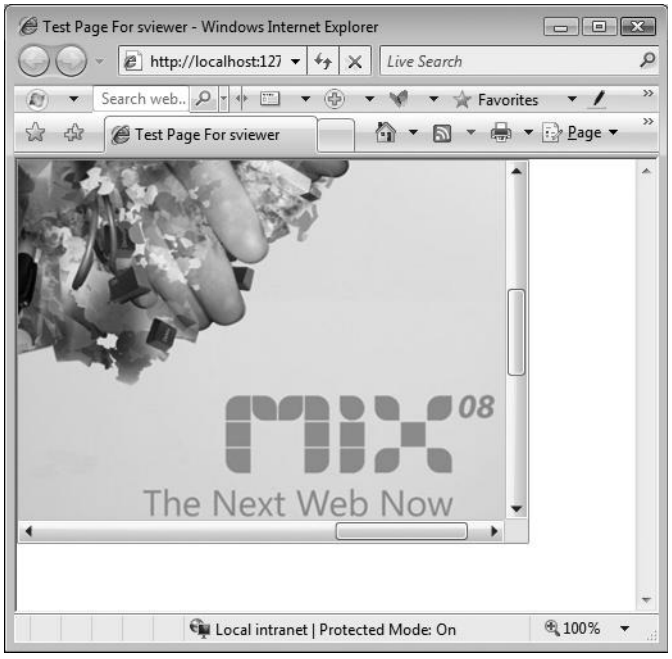

РИС. 9-9 Использование *ScrollViewer* с горизонтальной полосой прокрутки.

Подсказка Можно также переопределить видимость вертикальной полосы прокрутки. По умолчанию она видимая, но ее можно отключить (т.е. она не будет отображаться), задав свойству *VerticalScrollBarVisibility* значение *Hidden*.

Элемент управления *Slider*

С помощью элемента управления *Slider* пользователь может выбирать значение из заданного диапазона, передвигая вертикальный элемент (называемый *бегунком* (*thumb*)) вдоль линии, начиная от наименьшего значения в диапазоне, заканчивая наибольшим значением. Доступ к значению обеспечивается через свойство *Value* (Значение), возвращающее величину типа *Double*.

Элемент управления *Slider* можно настроить множеством способов. Например, свойства *Minimum* (Минимум) и *Maximum* (Максимум) используются для определения минимального и максимального значений диапазона. Если необходимо, чтобы пользователь выбирал значение из диапазона чисел от 0 до 100, задайте свойству *Minimum* значение 0 и свойству *Maximum* значение 100.

Еще одно свойство, связанное с элементом управления *Slider*, – *Orientation* (Ориентация), которое используется для определения направления перемещения бегунка. Ему может быть заданы значения *Horizontal* (Горизонтально) или *Vertical* (Вертикально).

Когда свойству *Orientation* задано значение *Horizontal*, по умолчанию перемещение *Slider* от наименьшего к наибольшему значению реализуется в направлении слева направо, и снизу вверх, если *Orientation* задано значение *Vertical*. Эти стандартные настройки можно переопределить свойством *IsDirectionReversed* (Обратное направление), задание значения *True* которому обеспечит изменение направления возрастания значений в *Slider*.

Кроме того, можно перемещать бегунок, щелкая на полосе перемещения типового элемента управления *Slider*. Каждый щелчок обеспечивает изменение значения на определенную величину. Это поведение определяется свойством *LargeChange* (Быстрое перемещение), задание числового значения, которому обеспечивает увеличение или уменьшение значения бегунка на заданную величину.

Далее приведен пример элемента управления *Slider* в XAML:

```
<Slider x:Name="sldr" Maximum="100"
Orientation="Vertical"
IsDirectionReversed="True"
LargeChange="10">
</Slider>
```

Здесь создается вертикальный *Slider* с наименьшим значением сверху, наибольшим – снизу. Он обеспечивает возможность, перетягивая бегунок, выбирать значения в диапазоне от 0 до 100. Также пользователь может увеличивать значения на 10, щелкая полосу перемещения.

Заключение

В данной главе продолжается изучение набора элементов управления Silverlight. Много внимания было уделено *DataGrid*, его свойствам, методам и событиям, поскольку это важный элемент управления, обеспечивающий приложениям функциональность, аналогичную электронным таблицам. При обсуждении *DataGrid* представлен код, предлагаемый Silverlight для упрощения привязки данных, а также рассмотрены шаблоны данных и их использование для компоновки таблиц. Кроме того, вы познакомились с другими расширенными инструментами, доступными в Silverlight, включая элементы управления *Calendar*, *DatePicker*, *ScrollViewer* и *Slider*.

На этом мы завершаем обзор элементов управления Silverlight и в следующей главе займемся созданием собственных элементов управления.

Глава 10

Мультимедиа в Silverlight: видео

В предыдущих главах вы познакомились с основными элементами управления, используемыми для построения насыщенных интерактивных Интернет-приложений. В данной главе будет рассмотрена работа с видеоданными и представлены возможные варианты, доступные для создания видеоприложений в Microsoft Silverlight.

Сначала подробно остановимся на элементе управления *MediaElement*, который обеспечивает API для управления аудио- и видеоданными, позволяя тем самым создавать замечательные мультимедийные приложения. Затем рассмотрим архитектуру и реализацию Silverlight-клиента для мультимедиа, защищенного технологией DRM (Digital Rights Management – управление правами на цифровые материалы). Также создадим пример, работающий с защищенным содержимым, размещенным в Интернете.

Элемент управления *MediaElement*

Одно из наиболее важных применений Silverlight в Веб – обеспечение поддержки мультимедиа следующего поколения, которое может воспроизводиться на разных платформах. Для этого Silverlight поддерживает элемент управления *MediaElement*, которому посвящен данный раздел. В нем поэтапно рассматривается процесс создания простого медиаплеера, обеспечивающего возможность прогрессивной загрузки и воспроизведения видеоданных. Кроме того, вы научитесь закрашивать поверхности с помощью видео-кисти, что позволит добавлять интересные графические эффекты. Элемент управления *MediaElement* поддерживает следующие форматы:

Видео

- WMV1: Windows Media Video 7
- WMV2: Windows Media Video 8
- WMV3: Windows Media Video 9
- WMVA: Windows Media Video Advanced Profile, не–VC-1
- WMVC1: Windows Media Video Advanced Profile, VC-1
- H.264: Видео, кодированное в популярном формате H264. *Обратите внимание, это нововведение Silverlight 3*.

Аудио

- WMA7: Windows Media Audio 7
- WMA8: Windows Media Audio 8
- WMA9: Windows Media Audio 9
- WMA10: Windows Media Audio 10
- MP3: ISO/MPEG Layer 3
- Моно или стерео
- Частоты дискретизации от 8 до 48 кГц

- Скорости передачи двоичных данных от 8 до 320 Кбит/с
- Переменная скорость передачи двоичных данных

Кроме этих форматов, элемент управления *MediaElement* также поддерживает списки воспроизведения ASX и протоколы *HTTP, HTTPS* и *MMS*.

Если говорить о потоковой передаче видео и/или аудио, *MediaElement* поддерживает потоковое вещание и потоковую передачу по запросу от сервера, выполняющего Windows Media. Если для URI задан протокол *MMS, потоковая передача включена; в противном случае, загрузка и воспроизведение файла осуществляются путем прогрессивной загрузки, при которой воспроизведение начинается только после заполнения буфера воспроизведения и затем эти два процесса выполняются параллельно*.

Если определен протокол *HTTP* или *HTTPS, происходит обратное. MediaElement сначала делает попытку выполнить прогрессивную загрузку, и уже в случае неудачи пытается применить потоковую передачу файла*.

Использование элемента управления *MediaElement*

Элемент управления *MediaElement* в базовой конфигурации настроить очень просто, но он имеет множество расширенных функций, освоив которые, вы сможете реализовывать довольно интересные сценарии. Но, прежде чем бегать, надо научиться ходить, поэтому сначала обратимся к решению самых обычных задач с помощью *MediaElement*.

Простое воспроизведение видео с помощью элемента управления *MediaElement*

Чтобы начать работу с элементом управления *MediaElement*, добавьте его на свою страницу и задайте в качестве значения его атрибута *Source* URL видеофайл, который хотите воспроизвести. Рассмотрим пример XAML-кода:

```
<UserControl
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        x:Class="SilverlightApplication5.MainPage"
        Width="640" Height="480">
        <Grid x:Name="LayoutRoot" Background="White">
                <MediaElement Source="/balls.wmv" />
        </Grid>
</UserControl>
```

Это обеспечит автоматическую загрузку и воспроизведение видеоданных. *Размер* области воспроизведения мультимедиа определяется следующими правилами:

- Если заданы свойства *Height* и *Width* элемента управления *MediaElement*, *MediaElement* использует их.
- Если задано одно из этих свойств, элемент управления *MediaElement* растягивает изображение, сохраняя пропорции.
- Если не заданы ни *Height*, ни *Width*, элемент управления *MediaElement* растягивает изображение до размеров родительского контейнера, сохраняя пропорции.
- Если заданы оба свойства и они превышают размер области просмотра элемента управления Silverlight, *MediaElement* обрезает видео соответственно размеру области просмотра.

Рассмотрим пример. Используемая в данной главе (и доступная для скачивания на Веб-сайте компании) видеозапись balls.wmv имеет размер 480 × 360. Если указать *MediaElement* воспроизводить

это видео и не задать его *Height* и *Width*, воспроизводимое видеоизображение будет автоматически растянуто с сохранением пропорций до размеров родительского контейнера. Если для *UserControl* выставить размер 200 × 200 и размер видео – 480 × 360, то на экране будет видна только часть видеоизображения размером 200 × 200 пикселов, начиная от верхнего левого угла. Обрезанная таким образом часть видео представлена на рис. 10-1.

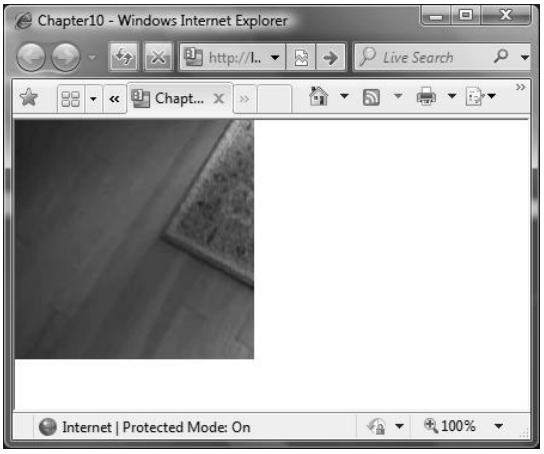

РИС. 10-1 Видеоизображение, обрезанное по размеру элемента управления Silverlight.

Размеры элемента управления *MediaElement*

Как было показано в предыдущем разделе, размер *MediaElement* важен при воспроизведении видео.

Управление шириной и высотой самого *MediaElement* осуществляется через его свойства *Height* и *Width*. При формировании визуального представления элемента управления, мультимедиа будет растягиваться (или сжиматься) так, чтобы поместиться в элемент управления. Если заданный размер элемента управления мультимедиа больше элемента управления Silverlight, изображение будет обрезано размерами элемента управления Silverlight.

Далее представлен пример элемента управления *MediaElement*, для которого заданы размеры 200 × 200. На рис. 10-2 можно увидеть, как будет отображаться видео:

```
<MediaElement Source="/balls.wmv" Height="200" Width="200" />
```

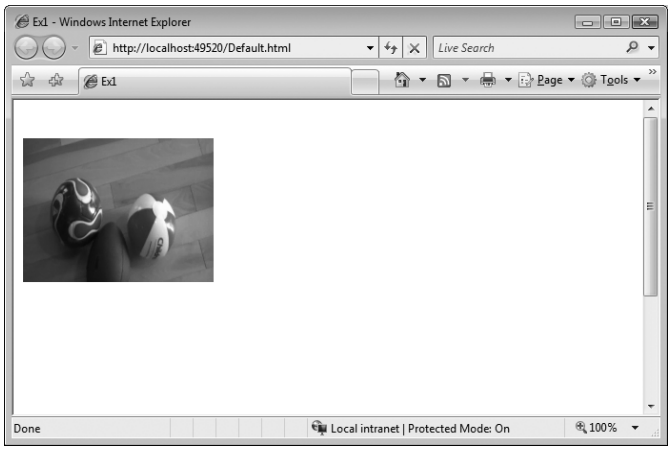

РИС. 10-2 Отображение видео после задания размеров *MediaElement*.

Растяжение мультимедиа

В предыдущем примере видеоизображение сворачивалось до размера элемента управления *MediaElement* (200 × 200). Как видно на рис. 10-2, размер видеоизображения (исходный размер которого, как вы помните, 480 × 360 пикселов) меняется так, чтобы оно поместилось в *MediaElement*,

но с сохранением его пропорций. Это может привести к появлению полей вверху и внизу видеоизображения, создавая эффект просмотра широкоформатного видео. Переопределить это поведение можно с помощью свойства *Stretch* элемента управления *MediaElement*. Это свойство принимает четыре разных значения:

- ***None*** Растяжения нет. Если *MediaElement* больше размера видеоизображения, видеоизображение отображается в центре области отображения. Если *MediaElement* меньше, будет видна центральная часть видеоизображения. Например, возьмем видео размером 480 × 360. Если размер *MediaElement* – 200 × 200, и *Stretch* задано значение *None*, отображается центральная область видеоизображения размером 200 × 200, как показано на рис. 10-3.

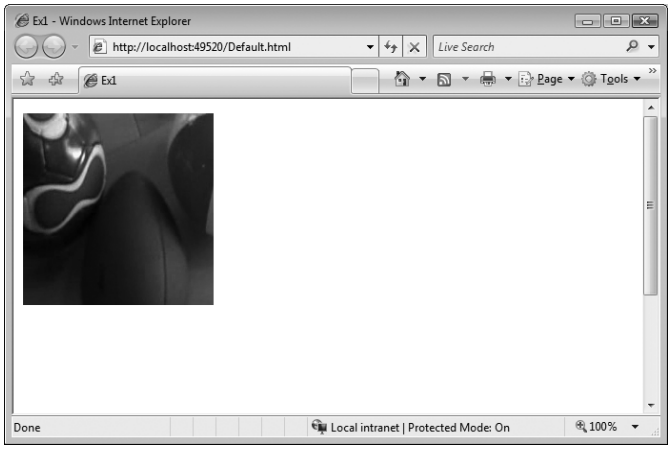

РИС. 10-3 Эффект задания свойству *Stretch* значения *None*.

- ***Uniform*** Это режим растяжения по умолчанию, он обеспечивает сохранение пропорций видеоизображения и добавление полей вверху, внизу или по бокам в случае необходимости.
- ***UniformToFill*** Это значение обеспечивает растяжение видео с сохранением его пропорций. При этом часть видеоизображения, не поместившаяся в область отображения, обрезается. Так, например, если ширина видео больше его высоты (например, 480 × 360) и оно должно отображаться в окне 200 × 200, видеоизображение будет обрезано, чтобы поместиться в окне просмотра (до меньшего квадрата, в данном случае). Результат этого можно увидеть на рис. 10-4. Если сравнить его с рис. 10-2, можно заметить, что видео обрезано.

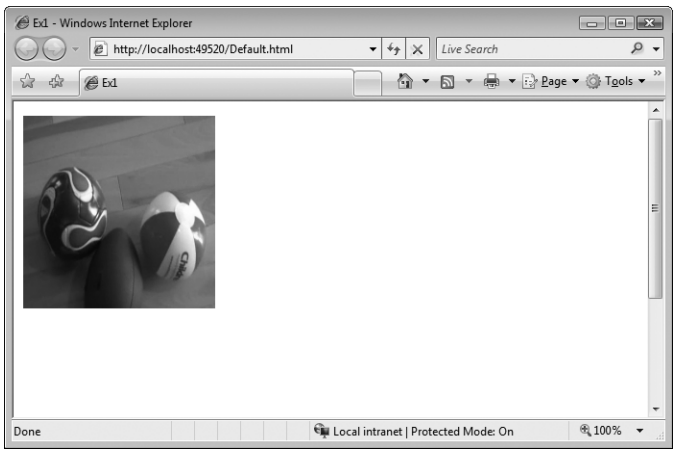

РИС. 10-4 Использование режима растяжения *UniformToFill*.

- ***Fill*** Это режим растяжения обеспечивает заполнение *MediaElement* видеоизображением с изменением его пропорций в случае необходимости. На рис. 10-5 показано отображение видео, когда свойству *Stretch* задано значение *Fill*. Как видно в этом случае, для заполнения области просмотра видеоизображение растянуто по вертикали.

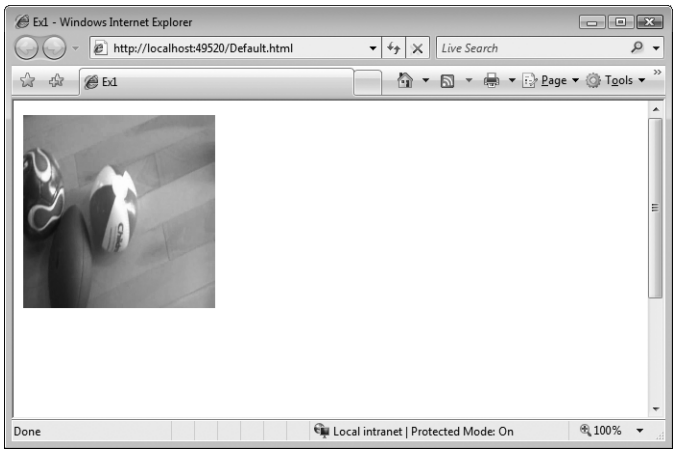

РИС. 10-5 Использование значения *Fill* свойства *Stretch*.

Прозрачность видеоизображения

Регулировать прозрачность *MediaElement* позволяет свойство *Opacity*. Это нормализованное свойство, т.е. значение 0 эквивалентно полной прозрачности, 1 – полной непрозрачности, все остальные значения в этом диапазоне представляют разные степени прозрачности. Видеоизображение отображается с заданной прозрачностью, при этом элементы, находящиеся позади элемента мультимедиа на экране, видны.

Далее представлен пример XAML, обеспечивающий отображение красного прямоугольника и *MediaElement*. Для *MediaElement* задана прозрачность 0,5, что делает видеоизображение полупрозрачным. Поскольку визуальное представление *MediaElement* формируется вторым, он размещается ближе в направлении по оси Z и располагается поверх прямоугольника. На рис. 10-6 можно увидеть результат выполнения этого примера.

```
<Rectangle Fill="Red" Height="100" Width="200" />
<MediaElement Source="/balls.wmv" Height="200"
      Width="200" Stretch="Fill" Opacity="0.5" />
```

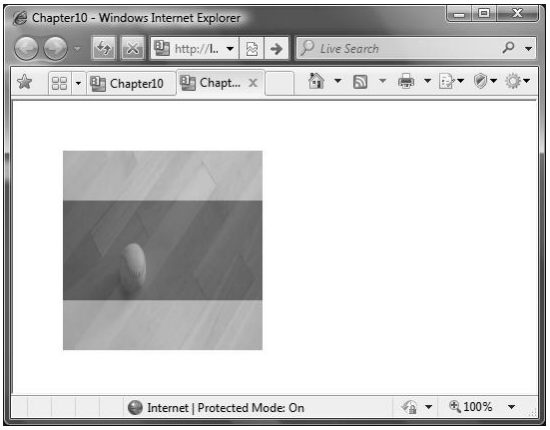

РИС. 10-6 Применение свойства *Opacity* для видеоизображения.

Использование трансформаций с элементом управления *MediaElement*

Подробно трансформации рассматриваются в Главе 5, «XAML: трансформация и анимация». Здесь хотелось бы отметить лишь то, что трансформации могут применяться и к элементу управления *MediaElement*. При этом отображаемое видео также трансформируется. Это может привести к созданию некоторых очень интересных эффектов. Например, далее представлен *MediaElement*, к которому применена трансформация наклонение:

```
<MediaElement Source="/balls.wmv" Height="200" Width="200" Stretch="Fill" >
  <MediaElement.RenderTransform>
    <SkewTransform AngleX="45"/>
  </MediaElement.RenderTransform>
</MediaElement>
```

Как это выглядит, можно увидеть на рис. 10-7.

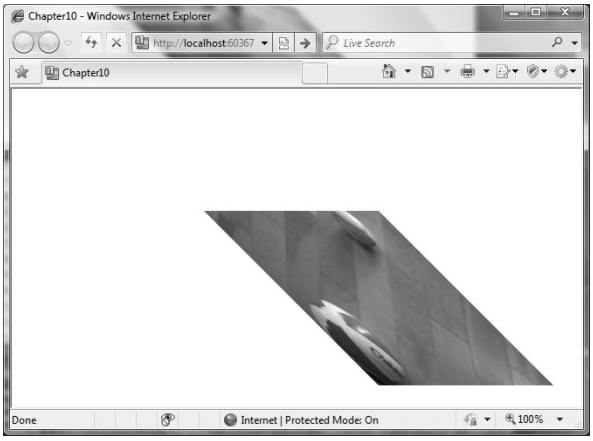

РИС. 10-7 Наклонение видео с помощью *SkewTransform*.

Расположение содержимого поверх видеоизображения

Silverlight позволяет размещать содержимое, включая текст и графические элементы, поверх видеоизображения. Это можно сделать либо с помощью свойств *ZOrder* (Z-порядок) элементов пользовательского интерфейса (UI), либо просто размещать элементы UI в том же месте, что и элемент мультимедиа, и объявлять их в XAML после него (более подробно UI рассматривается в Главе 1, «Введение в Silverlight 3»). Далее представлен пример *MediaElement*, и *Canvas* с прямоугольником и текстовым блоком, который перекрывает видеоизображение:

```
<MediaElement Source="/balls.wmv" Height="200" Width="200" Stretch="Fill" />
<Canvas Margin="20,140,0,0">
  <Rectangle Fill="Red" Height="40" Width="160" />
  <TextBlock>Subtitle on Video</TextBlock>
</Canvas>
```

На рис. 10-8 показано, как это будет выглядеть на экране.

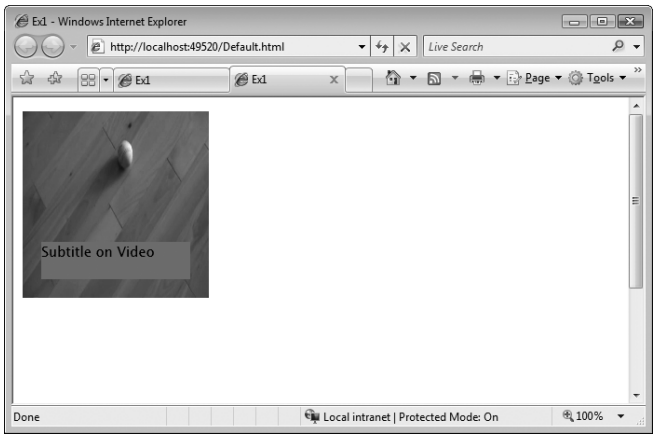

РИС. 10-8 Отображение содержимого поверх видео.

Вырезание мультимедиа геометрическими формами

В Главе 4, «Основы Silverlight XAML» рассматриваются вырезание и геометрические элементы. Эти возможности также могут применяться к *MediaElement*, для которого с помощью фигур или контуров

можно определить контур вырезания. Например, следующий XAML в качестве контура вырезания для *MediaElement* определяет эллипс:

```
<MediaElement Source="/balls.wmv" Height="200" Width="200" Stretch="Fill" >
  <MediaElement.Clip>
    <EllipseGeometry RadiusX="100" RadiusY="75" Center="100,75"/>
  </MediaElement.Clip>
</MediaElement>
```

Результаты представлены на рис. 10-9.

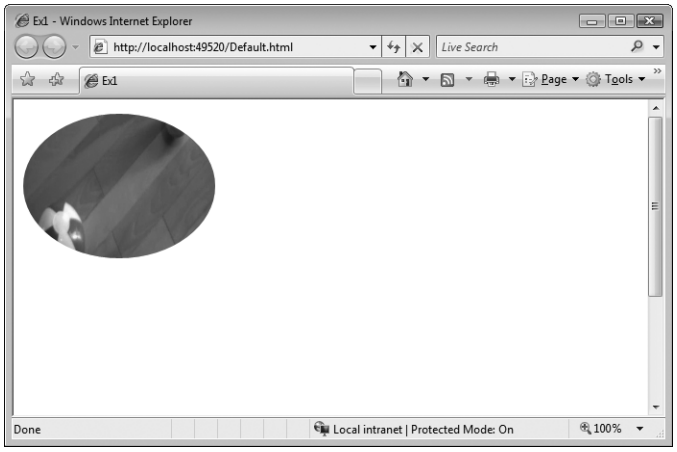

РИС. 10-9 Вырезание мультимедиа геометрическими формами.

Автоматическое воспроизведение

По умолчанию, если задан источник, *MediaElement* автоматически воспроизводит мультимедиа. Этим поведением можно управлять посредством свойства *AutoPlay* (Автоматическое воспроизведение). По умолчанию его значение *true*, но его можно переопределить и задать *false*. Можно воспроизвести мультимедиа позже, используя метод *Play* (Воспроизвести). Этот и другие методы и события, используемые при программировании *MediaElement*, рассматриваются в следующем разделе.

Управление воспроизведением звука

С помощью свойства *IsMuted* (Без звука) элемента управления *MediaElement* можно определить, будет ли аудиозапись сопровождать воспроизведение или нет. Это свойство Булевого типа, и если задать ему значение *true*, звук будет отключен.

Кроме того, с помощью свойства *Volume* (Громкость) можно управлять громкостью звука. Оно принимает нормализованное значение, где 0 эквивалентно отсутствию звука, 1 – полной громкости, и промежуточные значения представляют относительные уровни громкости. Итак, например, значение 0,43 обозначало бы воспроизведение звука с громкостью, составляющей 43% от полной громкости.

Наконец, балансом звука можно управлять с помощью свойства *Balance* (Баланс), которое принимает значения в диапазоне от -1 до +1. Значение -1 приводит к смещению баланса звука полностью влево, т.е. левый динамик воспроизводит звук со 100% громкостью, и в правом динамике будет 0% громкости. Значение +1 обеспечивает прямо противоположный эффект – баланс звука полностью смещается вправо, при этом правый динамик воспроизводит звук со 100% громкостью. Значение 0 обеспечивает равномерное распределение громкости между двумя динамиками.

Например, свойству *Balance* задано значение 0,8, тогда правый динамик будет воспроизводить звук с 80% громкостью и левый динамик – с 20%. Если используется значение -0,8, левый динамик будет воспроизводить звук с громкостью 80%, и правый динамик – с громкостью 20%.

Далее представлен некоторый XAML, определяющий, что звук не выключен, общая громкость составляет 50% и что баланс звука смещен вправо:

```
<MediaElement x:Name="vid" Source="/balls.wmv" Height="200" Width="200"
    Stretch="Fill" IsMuted="False" Volume="0.5" Balance="0.8" />
```

Программирование *MediaElement*

MediaElement предлагает насыщенную модель программирования, которая позволяет управлять воспроизведением методами *Play*, *Stop* (Остановить) и *Pause* (Приостановить). Также вы имеете возможность взаимодействовать с видео, отслеживая процесс буферизации и загрузки, а также реагировать на маркеры, размещенные в видеофайле. Кроме того, можно обработать события для перехвата, такие как поведение мыши.

Базовые элементы управления воспроизведением

Основными методами управления видео являются *Play*, *Stop* и *Pause*. Если свойству *AutoPlay* элемента управления *MediaElement* задано значение *false*, для начала воспроизведения видео необходимо применить *Play*. Даже если *AutoPlay* задано значение *true*, и воспроизведение начинается автоматически, с помощью этих методов его можно остановить или приостановить. Далее представлен пример XAML, в котором имеется элемент мультимедиа и три простых элемента управления воспроизведением, реализованных как элементы *TextBlock*:

```
<MediaElement x:Name="vid" Source="/balls.wmv" Height="200" Width="200" Stretch="Fill" />
<Canvas Canvas.Top="160">
  <Rectangle Fill="Black" Width="200" Height="24" Opacity="0.7"/>
  <TextBlock Foreground="White" Canvas.Left="20">Play</TextBlock>
  <TextBlock Foreground="White" Canvas.Left="80">Stop</TextBlock>
  <TextBlock Foreground="White" Canvas.Left="140">Pause</TextBlock>
</Canvas>
```

На рис. 10-10 показано, как эти элементы управления будут выглядеть на видеоизображении.

Для создания элементов управления вы задаете имя функции, которая должна выполняться в ответ на событие мыши, используя атрибут самого элемента управления видео. Намного более подробно обработка событий Silverlight с помощью JavaScript рассматривается в Главе 7, «Механизм доступа к объектной модели браузера». Однако, в данном случае, просто требуется начать, остановить или приостановить воспроизведение видео, когда пользователь щелкнет соответствующий текстовый блок. Это реализуется путем обработки события *MouseLeftButtonDown* текстового блока, которое ассоциируется с функциями, воспроизводящими, приостанавливающими или останавливающими воспроизведение.

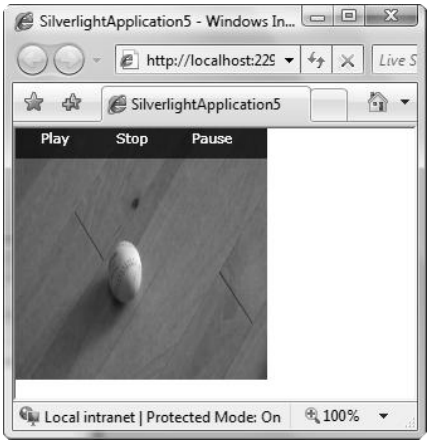

РИС. 10-10 Добавление элементов управления в видеоизображение.

В JavaScript-коде необходимо обеспечить, чтобы эти функции были доступны со страницы, на которой размещается элемент управления Silverlight. Итак, они могут быть реализованы либо путем использования элементов JavaScript на странице, либо в файле .js, который включен в страницу с помощью элемента JavaScript, свойству *Src* которого задано соответствующее значение.

Если для сведения воедино содержимого XAML используется Microsoft Expression Blend 2, он обеспечивает для Page.xaml псевдо-файл выделенного кода Page.xaml.js. Это идеальное место для реализации функциональности JavaScript. При использовании более поздних версий Expression Blend JavaScript придется создавать вручную.

Создадим проект Silverlight 1.0 Site. Далее представлен XAML, описывающий похожий UI, как был создан в предыдущем примере, но с объявлениями обработчиков событий:

```
<Canvas
          xmlns="http://schemas.microsoft.com/client/2007"
          xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
          Width="640" Height="480"
          Background="White"
          x:Name="Page">
<MediaElement x:Name="vid" Source="balls.wmv"
           Height="200" Width="200" Stretch="Fill" />
<Canvas Canvas.Top="160">
  <Rectangle Fill="Black" Width="200" Height="24" Opacity="0.7"/>
  <TextBlock MouseLeftButtonDown="doPlay"
         Foreground="White"
         Canvas.Left="20">Play</TextBlock>
  <TextBlock MouseLeftButtonDown="doStop"
         Foreground="White"
         Canvas.Left="80">Stop</TextBlock>
  <TextBlock MouseLeftButtonDown="doPause"
         Foreground="White"
         Canvas.Left="140">Pause</TextBlock>
</Canvas>
</Canvas>
```

Теперь можно написать кода на JavaScript для начала, остановки и приостановки воспроизведения. Вот этот код:

```
function doPlay(sender, args)
{
  var meVid = sender.findName("vid");
  meVid.Play();
}

function doStop(sender, args)
{
  var meVid = sender.findName("vid");
  meVid.Stop();
}

function doPause(sender, args)
{
  var meVid = sender.findName("vid");
  meVid.Pause();
}
```

Когда функция JavaScript определяется как обработчик события, она должна принимать два параметра. Первый, *sender*, - это объект, который формирует событие. Второй, *args*, содержит аргументы, которые включаются как часть события.

Теперь в этой JavaScript-функции необходимо получить ссылку на объект *MediaElement*, управление которым осуществляется. Выполняется это с помощью метода *findName* объекта *sender*. Хотя отправителем является текстовый блок, который щелкнул пользователь, выполнение его метода *findName* все равно обеспечит выполнение поиска по всему XAML-документу до тех пор, пока не будет найден элемент *vid* (который был создан путем применения атрибута *x:Name* к нашему объекту

MediaElement). Если вернуться к XAML, можно увидеть, что в *MediaElement* был элемент *x:Name* со значением *vid*, так что поиск должен быть успешным. Ссылка на соответствующий объект *MediaElement* будет сохранена в переменной JavaScript *meVid*. Теперь для управления воспроизведением осталось просто вызывать методы *Play*, *Stop* или *Pause* для этого объекта.

Если проект приложения для Silverlight версии 2 и 3 создается с помощью Microsoft .NET Framework, все даже еще проще[1]: не надо использовать *findName* для доступа к элементу, и ему уже присвоено имя *vid*, так что код будет выглядеть так:

```
private void doPlay(object sender, MouseButtonEventArgs e)
{
  vid.Play();
}
private void doPause(object sender, MouseButtonEventArgs e)
{
  vid.Pause();
}


private void doStop(object sender, MouseButtonEventArgs e)
{
  vid.Stop();
}
```

Управление буферизацией и загрузкой

При использовании прогрессивной загрузки видео подсистема мультимедиа определяет, сколько видеоданных необходимо кэшировать, прежде чем будет возможно воспроизведение. В зависимости от того, какая полоса пропускания необходима для загрузки видео, и какая полоса пропускания доступна, инфраструктура мультимедиа создает буфер для размещения объема видеоданных, достаточного для начала воспроизведения, в ходе которого загрузка видео продолжается в фоновом режиме.

Когда буфер полон на 100%, начинается воспроизведение. Воспроизведение может прерываться при изменении состояния связи и дозагрузки буфера. Silverlight позволяет отслеживать это поведение с помощью события *BufferingProgressChanged* (Показатель процесса буферизации изменился) и свойства *BufferingProgress* (Показатель процесса буферизации). Используя эти событие и свойство, можно показывать пользователям текущее состояние буферизации или выполнять логику для улучшения взаимодействия с пользователем (UX) как результат условий буферизации. Например, в условиях плохого соединения буферизация может не превышать 50%. Вы можете перехватить это значение и предоставить соответствующую обратную связь своему клиенту.

Для управления буферизацией задайте в *MediaElement* обработчик событий, определяющий функцию для обработки события *BufferingProgressChanged*:

```
<MediaElement x:Name="vid" Source="/balls.wmv" Height="200" Width="200"
      Stretch="Fill" BufferingProgressChanged="doBuff"/>
<TextBlock x:Name="txtBuff"></TextBlock>
```

Таким образом, функция *doBuff* будет выполняться каждый раз при изменении наполненности буфера. Это событие тесно связано со свойством *BufferingProgress*. Данное свойство принимает значения в диапазоне от 0 до 1, где 0 соответствует пустому буферу, и 1 – полному. Событие формируется при изменении наполнения буфера на 5% (т.е. 0,05) или более и когда буфер заполнен полностью.

Далее представлен код, который может использоваться для ответа на это событие и обеспечения обратной связи о текущем состоянии буфера пользователям.

---

[1] Не забудьте объявить обработчики событий для элементов управления TextBlock (*прим. редактора*)

Сначала JavaScript-версия:

```
function doBuff(sender, args)
{
  var theText = sender.findName("txtBuff");
  var meVid = sender.findName("vid");
  var prog = meVid.BufferingProgress * 100;
  prog = "Buffering % " + prog;
  theText.Text = prog;
}
```

И вот, как это было бы сделано в выделенном коде .NET (на C#):

```
private void doBuff(object sender, RoutedEventArgs e)
{
 double prog = vid.BufferingProgress * 100;
 txtBuff.Text = "Buffering % " + prog;
}
```

Если у вас не видно текст с отображением буферизации в Silverlight-приложении, обратите внимание на используемый элемент компоновки. Если вы использовали <*Grid*>, текст может находиться под «кнопками» *Play*, *Stop* и *Pause*. *В этом случае* задайте для текстовых блоков небольшой отступ с помощью свойства *Margin*.

Можно переопределить автоматическую буферизацию, задавая конкретную продолжительность буферизованного видео. Например, если необходимо всегда иметь 10-секундный буфер видео и, таким образом, снизить риск прерывания воспроизведения из-за повторной синхронизации буфера при ухудшении состояния связи, можно задать свойство *BufferingTime* (Размер буфера). В качестве его значения задается временной диапазон. Так, например, чтобы задать 10-секундный буфер, *BufferingTime* задается значение 0:0:10, как показано в следующем примере:

```
<MediaElement x:Name="vid" Source="/balls.wmv"
    Height="200" Width="200" Stretch="Fill"
    BufferingProgressChanged="doBuff"
    BufferingTime="0:0:10"/>


<TextBlock x:Name="txtBuff"></TextBlock>
<TextBlock x:Name="txtDown"></TextBlock>
```

Если прогрессивная загрузка недоступна или не поддерживается, придется загружать весь видеофайл целиком и только потом начинать его воспроизведение. В этом случае событие *DownloadProgressChanged* и свойство *DownloadProgress* могут использоваться для получения состояния загрузки. Принципы работы с ними аналогичны функциям буферизации. Далее представлен XAML, определяющий событие *DownloadProgressChanged*:

```
<MediaElement x:Name="vid" Source="/balls.wmv"
    Height="200" Width="200" Stretch="Fill"
    BufferingProgressChanged="doBuff"
    BufferingTime="0:0:10"
    DownloadProgressChanged="doDown"/>

<TextBlock x:Name="txtBuff"></TextBlock>
<TextBlock x:Name="txtDown"></TextBlock>
```

И теперь код функции *doDown* , обрабатывающей это событие. Опять же, сначала рассмотрим JavaScript-версию:

```
function doDown(sender, args)
{
  var theText = sender.findName("txtDown");
  var meVid = sender.findName("vid");
  var prog = meVid.DownloadProgress * 100;
  prog = "Downloading % " + prog;
  txtDown.Text = prog;
}
```

И вот .NET-версия:

```
private void doDown(object sender, RoutedEventArgs e)
{
  double prog = vid.DownloadProgress * 100;
  txtDown.Text = "Downloading % " + prog;

}
```

Управление текущим состоянием видео

Silverlight представляет свойство *CurrentState* (Текущее состояние) и связанное с ним событие *CurrentStateChanged* (Текущее состояние изменилось), которое может использоваться для ответа на изменение состояния мультимедиа.

Действительными состояниями свойства *CurrentState* являются:

- ***Buffering (Буферизация)*** Буфер не заполнен на 100%, поэтому воспроизведение приостановлено до полного заполнения буфера.
- ***Closed (Закрыто)*** Мультимедиа закрыто.
- ***Error (Ошибка)*** Возникла проблема с загрузкой, буферизацией или воспроизведением мультимедиа.
- ***Opening (Открывается)*** Мультимедийный файл найден, и буферизация или загрузка должны вот-вот начаться.
- ***Paused (Приостановлено)*** Воспроизведение приостановлено.
- ***Playing (Воспроизводится)*** Идет воспроизведение мультимедиа.
- ***Stopped (Остановлено)*** Воспроизведение остановлено.
- ***Individualizing (Выполняется индивидуализация)*** Выполняется запрос к серверу системы индивидуализации для управления правами DRM. Этот вопрос рассматривается далее в этой главе.
- ***Acquiring License (Запрос лицензии)*** Silveright связывается с сервером системы лицензирования для управления правами DRM с целью получения лицензии на воспроизведение защищенного содержимого. Этот вопрос рассматривается далее в этой главе.

Вот как событие *CurrentStateChanged* элемента управления *MediaElement* определяется в XAML:

```
<MediaElement x:Name="vid" Source="/balls.wmv" Height="200" Width="200"
   Stretch="Fill" CurrentStateChanged="doState" BufferingTime="0:0:10" />
```

В данном фрагменте кода задается функция *doState*, которая будет вызываться в ответ на изменение текущего состояния. Далее представлен пример такой JavaScript-функции, которая использует значение свойства *CurrentState* элемента управления *MediaElement* в окне уведомления:

```
function doState(sender, args)
{
  var meVid = sender.findName("vid");
  alert(meVid.CurrentState);
}
```

В C#, вместо окна уведомления *MessageBox*, для вывода содержимого свойства *CurrentState* на экран можно использовать *TextBlock*:

```
private void doState(object sender, RoutedEventArgs e)
{
  txtBuff.Text = vid.CurrentState.ToString();
}
```

Управление положением курсора воспроизведения

Для управления текущим положением курсора воспроизведения используются свойства *NaturalDuration* (Номинальная продолжительность) и *Position* (Положение) элемента мультимедиа. После того, как свойство *CurrentState* устанавливается в значение *Opened*, становится возможным получение значения и свойства *NaturalDuration*. Продолжительность видео в секундах можно получить с помощью *NaturalDuration.Seconds* и потом преобразовать это значение в часы, минуты и секунды.

В этом примере событие *CurrentStateChanged* элемента управления *MediaElement* ассоциировано с функцией *doState* (из предыдущего примера). Однако теперь эта функция перехватывает значение свойства *NaturalDuration*. В JavaScript-версии используется вспомогательная функция *convertDT* для преобразования возвращаемого значения в строку. Рассмотрим код на JavaScript:

```
function doState(sender, args)
{
  var meVid = sender.findName("vid");
  var txtStat = sender.findName("txtStat");
  var datetime = new Date(0, 0, 0, 0, 0, meVid.naturalDuration.Seconds)
  durationString = convertDT(datetime);
  txtStat.Text = durationString.toString();
}

function convertDT(datetime)
{
  var hours = datetime.getHours();
  var minutes = datetime.getMinutes();
  var seconds = datetime.getSeconds();
  if (seconds < 10) {
     seconds = "0" + seconds;
  }

  if (minutes < 10) {
     minutes = "0" + minutes;
  }

  var durationString;
  if (hours > 0) {
     durationString = hours.toString() + ":" + minutes + ":" + seconds;
  }
  else {
    durationString = minutes + ":" + seconds;
  }
  return durationString;
}
```

Далее следует замечательный пример того, как .NET Framework в Silverlight 3 существенно упрощает разработку: все это может быть реализовано одной строкой кода на C#:

```
private void doState(object sender, RoutedEventArgs e)
   {
     txtStat.Text = vid.NaturalDuration.ToString();
   }
```

Вот XAML, в котором используется представленная выше функция:

```
<Canvas x:Name="sample9" Opacity="1">
  <MediaElement x:Name="vid" Source="/balls.wmv" Height="200" Width="200"
     Stretch="Fill" CurrentStateChanged="doState"
     BufferingTime="0:0:10" />
  <TextBlock x:Name="txtStat"></TextBlock>
</Canvas>
```

Получить текущее положение курсора воспроизведения можно с помощью свойства *Position*. В этом примере позиция выводится в текстовом блоке при приостановке воспроизведения (Для тестирования можете оставить в XAML-коде три «кнопки» *Play*, *Stop* и *Pause* и соответствующие им обработчики

событий *doPlay* и *doStop* с предыдущего примера, но закомментировать тело функции *doState*). Необходимый код на JavaScript для функции *doPause* выглядит примерно так:

```
function doPause(sender, args)
{
  var meVid = sender.findName("vid");
  meVid.Pause();
  var txtStat = sender.findName("txtStat");
  var datetime = new Date(0,0,0,0,0, meVid.Position.Seconds);
  positionString = convertDT(datetime);
  txtStat.Text = positionString.toString();
}
```

И, как и ранее, эквивалентный код на C# намного проще:

```
private void doPause(object sender, MouseButtonEventArgs e)
   {
     vid.Pause();
     txtStat.Text = vid.Position.ToString();
   }
```

Использование маркеров временной шкалы мультимедиа

Маркер временной шкалы – это элемент метаданных, ассоциированный с конкретной точкой на временной шкале мультимедиа. Обычно маркеры создаются и кодируются в мультимедиа заранее с помощью такого ПО, как Expression Encoder, и часто применяются для установления в видео поисковых меток.

Silverlight поддерживает эти маркеры и формирует событие *MarkerReached* (Маркер достигнут) при встрече с маркером в ходе воспроизведения. Это событие можно перехватывать и использовать как триггер для действий, которые должны выполняться при достижении данной отметки.

Далее представлен XAML, в котором с помощью атрибута *MarkerReached* задается обработчик события встречи с маркером, JavaScript-функция *handleMarker*:

```
<MediaElement x:Name="vid" Source="/balls.wmv" Height="200" Width="200" MarkerReached="handleMarker" />
```

Среди аргументов данного события имеется объект *marker* (маркер). Этот объект содержит объект *TimeSpan* (Диапазон времени) с меткой времени, в которой установлен маркер. В предыдущем разделе, «Управление положением курсора воспроизведения», представлен пример преобразования объекта *TimeSpan* в формат, удобный для восприятия пользователем. Также *marker* содержит свойство *Type* (Тип). Его значением является строка, которую задает человек, кодирующий видео. Наконец, имеется параметр *Text*, допускающий в качестве своего значения произвольный текст и обычно используемый для описания параметра. Далее представлен JavaScript-код для чтения всех трех параметров и создания строки, отображаемой в окне уведомления:

```
function handleMarker(sender, args)
{
  var strMarkerStatus = args.marker.time.seconds.toString();
  strMarkerStatus += "  :  ";
  strMarkerStatus += args.marker.type;
  strMarkerStatus += "  :  ";
  strMarkerStatus += args.marker.text;
  alert(strMarkerStatus);
}
```

Код на C# очень похож на предыдущий; заметьте, что args типа *TimeLineMarkerRoutedEventArgs*:

```
private void handleMarker(object sender,
      TimelineMarkerRoutedEventArgs e)
{
 string strMarkerStatus = e.Marker.Time.ToString();
 strMarkerStatus += "  :  ";
```

```
  strMarkerStatus += e.Marker.Type;
  strMarkerStatus += " : ";
  strMarkerStatus += e.Marker.Text;
}
```

Как это выглядит, можно увидеть на рис. 10-11.

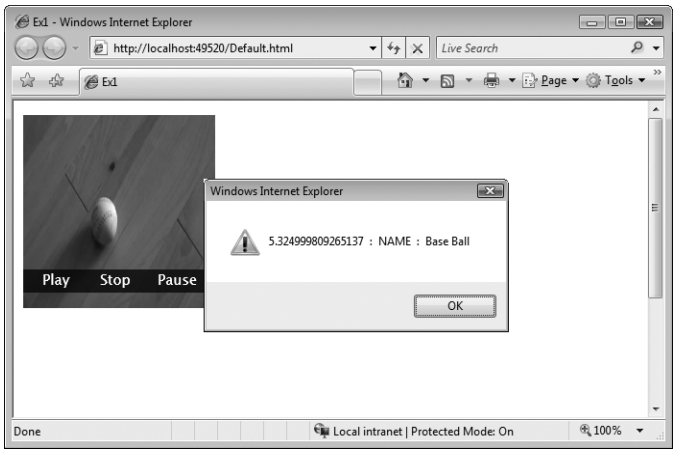

РИС. 10-11 Перехват маркеров временной шкалы в Silverlight.

Маркеры временной шкалы можно добавлять в медиафайл также динамически, используя код на Silverlight. При этом поисковые метки можно задавать через процентное соотношение положения метки к общей продолжительности файла, например.

Далее представлен пример, в котором XAML для *MediaElement* определяет функцию *handleOpen* (Обработка открытия), выполняющуюся при открытии файла мультимедиа. Она вставляет новый объект временной шкалы в видео на 10-секундной отметке. Этот элемент не сохраняется в видео постоянно, а только до завершения сеанса.

```
<MediaElement x:Name="vid" Source="/balls.wmv"
   Height="200" Width="200"
   MarkerReached="handleMarker"
   MediaOpened="handleOpened" />
```

Обработчик должен создать новый элемент временной шкалы в XAML и добавить его в коллекцию маркеров *MediaElement*. В данном случае, задается маркер временной шкалы на отметке 10 секунд, для которого *Type* определен как 'My Temp Marker' (мой временный маркер) и *Text* – 'Dynamically Added Marker' (Динамически добавляемый маркер).

Вот как это выглядело бы в JavaScript:

```
function handleOpened(sender, args)
{
  var marker =
  sender.getHost().content.createFromXaml(

    "<TimelineMarker Time='0:0:10'" +
    " Type='My Temp Marker' Text='Dynamically Added Marker' />");
  sender.markers.add(marker);
}
```

И вот аналогичный код в .NET:

```
private void handleOpened(object sender, RoutedEventArgs e)
{
  TimelineMarker t = new TimelineMarker();
  t.Time = new TimeSpan(0, 0, 0, 10);
  t.Type = "My Temp Marker";
  t.Text = "Dynamically Added Marker";
  vid.Markers.Add(t);
}
```

Коллекцию маркеров *MediaElement* составляют объекты *TimeLineMarker* (Маркер временной шкалы), поэтому для введения нового маркера просто создается новый объект *TimeLineMarker*, задаются его свойства *Time*, *Type* и *Text*, и этот объект добавляется в коллекцию. Silverlight имеет достаточно развитую логику и знает, когда должно быть сформировано событие, исходя из заданного *Time*, поэтому при создании новых маркеров нет необходимости добавлять их в порядке расположения на временной шкале.

Теперь, когда *MediaElement* достигает 10-секундной отметки при воспроизведении, появляется диалоговое окно уведомления, как показано на рис. 10-12.

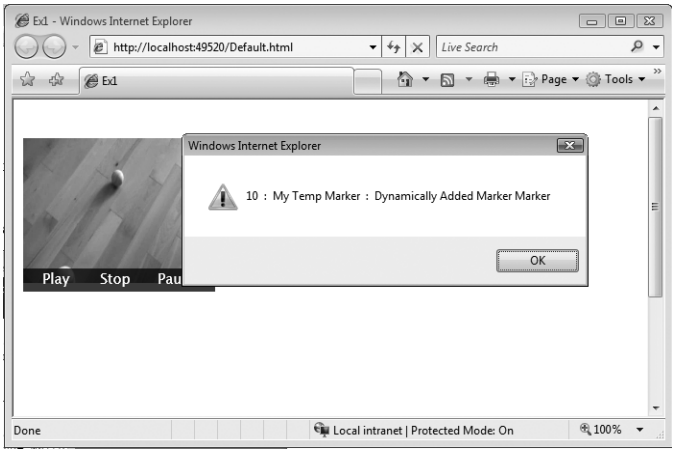

РИС. 10-12 Использование динамически добавляемого маркера.

Закрашивание поверхностей видеоизображением с помощью *VideoBrush*

Особенно замечательная возможность Silverlight – использование видео для заливки поверхностей с помощью *VideoBrush*. Сделать это очень просто. Во-первых, необходим элемент мультимедиа, который загрузит видео. Он должен быть скрыт и не должен принимать события мыши. Для этого его прозрачность задается равной 0 и свойству *IsHitTestVisible* (Объект виден для механизма обработки кликов) присваивается значение *false*. Также элементу мультимедиа должно быть задано имя посредством свойства *x:Name*. Рассмотрим пример:

```
<MediaElement x:Name="vid" Source="/balls.wmv" Opacity="0" IsHitTestVisible="False" />
```

Теперь объект *VideoBrush* можно применять к любому объекту точно так же, как любую другую кисть. В качестве источника кисти задается *MediaElement* (вот почему ему обязательно должно быть присвоено имя). Также можно задать свойство *Stretch* для управления визуальными эффектами кисти.

Например, здесь представлен *TextBlock*, для закрашивания *Foreground* которого используется *VideoBrush*:

```
<TextBlock FontFamily="Verdana" FontSize="80"
    FontWeight="Bold" TextWrapping="Wrap"
    Text="Video">
  <TextBlock.Foreground>
    <VideoBrush SourceName="vid"/>
  </TextBlock.Foreground>
</TextBlock>
```

Silverlight формирует визуальное представление текста, используя *VideoBrush*. Это продемонстрировано на рис. 10-13.

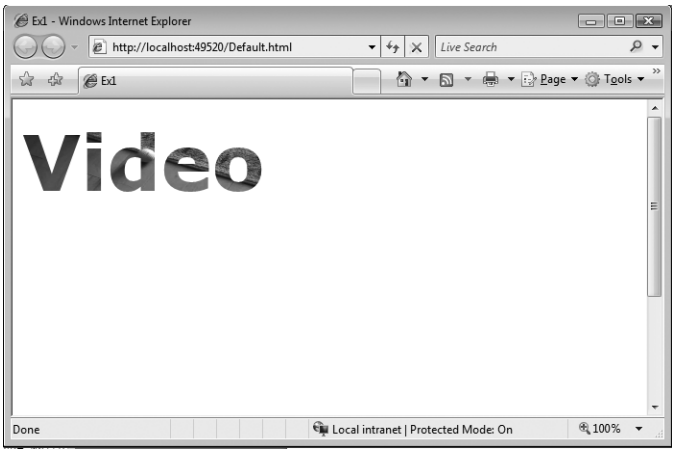

РИС. 10-13 Использование *VideoBrush* для закрашивания текста.

Поддержка видео в формате H.264

В Silverlight 3 добавлена поддержка декодера H.264, таким образом, <*MediaElement*> может обеспечивать воспроизведение в Silverlight содержимого, кодированного в формате H.264. Это знаменательное событие для компаний, вложивших средства в оцифровку своих ресурсов в этот формат и желающих использовать преимущества Silverlight для построения насыщенных пользовательских интерфейсов.

Если вы никогда ранее не работали с H.264, это можно сделать в полной версии Expression Encoder 3 или полной версии Expression Encoder 2, который при наличии пакета обновлений Service Pack 1 поддерживает кодирование в этот формат. Если у вас нет Expression Encoder 2 либо 3, бесплатный кодек можно найти по адресу *http://www.h264encoder.com*.

Далее в данном разделе будет рассматриваться кодирование файлов с использованием Expression Encoder 3.

В Expression Encoder можно импортировать файлы во многих форматах. Воспользуемся здесь файлом в формате MOD, который применяется во многих видеокамерах.

Запустим Expression Encoder и нажмем кнопку Import (Импортировать). Выберите свой файл, и Expression Encoder загрузит и подготовит его. Прежде чем выполнять кодирование, на вкладке Encode (Кодировать) можно выбрать используемый формат MP4 и профиль кодирования (рис. 10-14).

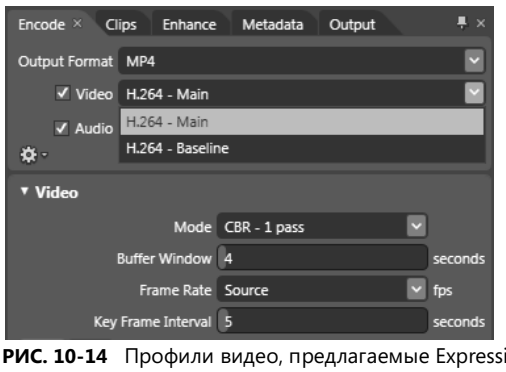

РИС. 10-14 Профили видео, предлагаемые Expression Encoder.

Как видите, для H.264 предлагается два профиля: Main (Основной) и Baseline (Базовый). Выберите основной.

Чтобы выбрать часть видеозаписи для кодирования (если требуется вырезать часть видео), перемещайте желтый маркер по временной шкале, располагающейся сразу под окном видео. Щелкните правой кнопкой мыши и выберите Mark In (Разметить до), чтобы обрезать все до этого

момента, или Mark Out (Разметить после), чтобы обрезать все после маркера. Когда все готово, нажимаем Encode (Кодировать) для запуска процесса кодирования.

Обратите внимание, что Expression Encoder 2 SP1 при кодировании в формат H.264 *не* поддерживает Silverlight. Эта поддержка есть для Expression Encoder 3..

По завершении кодирования получаем MP4-файл в формате H.264. Такой файл может быть воспроизведен с помощью Silverlight-элемента *MediaElement*.

Рассмотрим простой пример. Обратите внимание, что используется расширение файла .mp4:

```
<UserControl x:Class="SI3BladH264.MainPage"
   xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
   xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
   Width="640" Height="480">
   <Grid x:Name="LayoutRoot" Background="White">
      <MediaElement AutoPlay="True" Width="400" Height="400"
        Source="/MOV0AC.mp4"></MediaElement>
   </Grid>
</UserControl>
```

Как это выглядит, можно увидеть на рис. 10-15.

РИС. 10-15 Воспроизведение содержимого в формате H.264.

Защита мультимедиа с помощью управления правами на цифровые материалы

Silverlight совместим с технологией DRM, которая позволяет защищать видеоресурсы от несанкционированного копирования. Данная книга не рассматривает настройку *сервера* управления правами на цифровые материалы (DRM), подробную информацию можно найти на сайте Microsoft. Данный раздел является лишь очень кратким введением в эту сложную тему и показывает, как просто создать *клиент* для работы с защищенным содержимым.

В данном разделе будет рассмотрено, как применять DRM в Silverlight, а также представлен некоторый код клиента, работающего с защищенным DRM содержимым, которое размещено Microsoft.

Как выполняется DRM в Silverlight

DRM в Silverlight осуществляется через процесс, включающий пять этапов:

1. Silverlight-клиент выполняет доступ к содержимому путем задания свойства *Source MediaElement*,

как демонстрировалось ранее.

2. В случае применения прогрессивной загрузки Silverlight-клиент загружает содержимое. При потоковой передаче выполняется загрузка некоторой части содержимого и медиазаголовков.

3. Silverlight-клиент видит, что содержимое защищено, и связывается с *сервером системы индивидуализации*. Это сервис Microsoft, который предоставляет клиенту уникальный ID, что обеспечивает возможность его лицензирования.

4. Пройдя индивидуализацию, клиент связывается с сервером системы лицензирования для получения лицензии на воспроизведение содержимого.

5. Сервер системы лицензирования предоставляет лицензию, и Silverlight-клиент может воспроизводить содержимое.

Весь этот процесс отображен на рис. 10-16.

РИС. 10-16 Процесс доступа к защищенному DRM содержимому.

Silverlight поддерживает две формы расшифровки защищенного DRM содержимого: традиционное управление цифровыми правами Windows Media 10 (Windows Media Digital Rights Management 10, WMDRM 10) и PlayReady (Готовность к воспроизведению). Существующее содержимое, зашифрованное WMDRM, может воспроизводиться Silverlight, но все равно необходим сервер системы лицензирования PlayReady. Также обратите внимание, что для шифрования данных медиаисточников реального времени и прямой потоковой передачи должен использоваться WMDRM 10.

Если вы знакомы с использованием PlayReady для DRM-защиты, важно отметить, что Silverlight поддерживает не все возможности PlayReady, а только некоторые из них:

- Silverlight DRM поддерживает только форматы Windows Media Audio (WMA), Windows Media Video (WMV), и основанные на PlayReady PlayYourAudio (PYA) и PlayYourVideo (PYV).
- Silverlight доступен только один тип лицензии: временная лицензия на однократное воспроизведение.
- Нет функциональности кэширования лицензии.

Создание Silverlight-клиента для работы с защищенным DRM содержимым

Рассмотрение сервера с защищенным DRM содержимым выходит за рамки данной книги. Вы можете создать и протестировать клиент для работы с защищенным DRM содержимым, используя Silverlight.

Перейдем на сайт по адресу *http://web.sldrm.video.msn.com/d1/sldrm.html*. На этом сайте размещаются экземпляры одного фрагмента видео, кодированные с применением различных кодеков и доставляемые разными методами. Здесь также имеются версии, защищенные с помощью PlayReady, и не защищенные.

Этот сайт представлен на рис. 10-17.

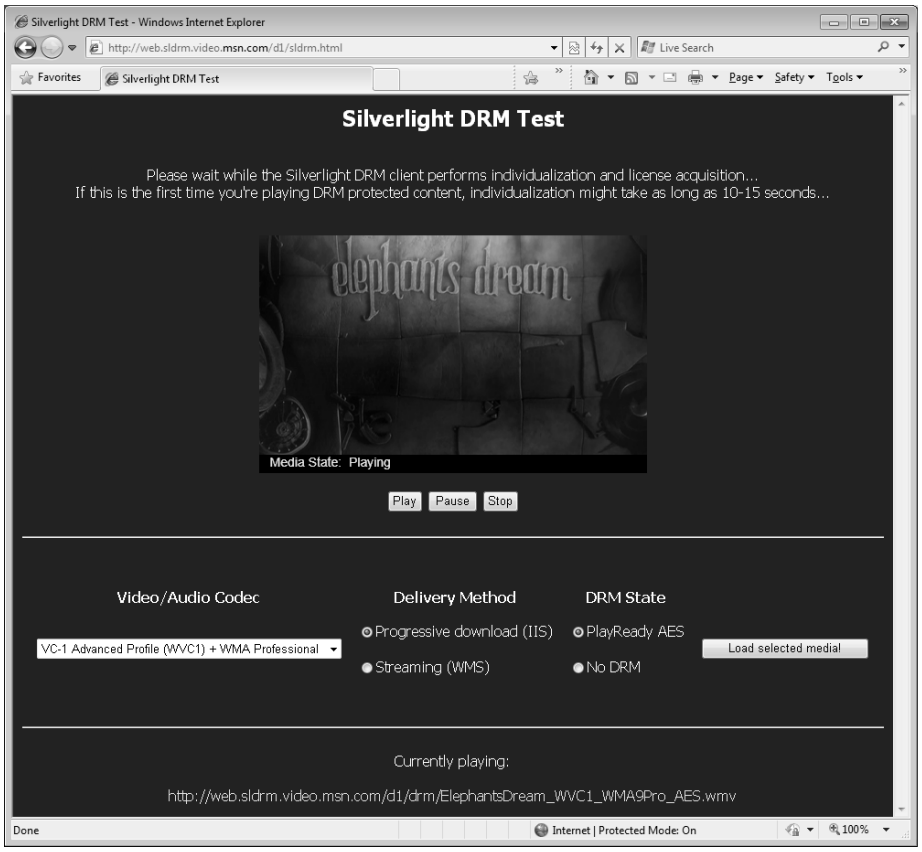

РИС. 10-17 Пример сайта DRM.

Как видите, на сайте предлагается выпадающий список, из которого можно выбрать необходимый видео- и/или аудиокодек, затем можно задать метод доставки и DRM-состояние и загрузить выбранный материал.

Также внизу страницы отображается адрес выбранного материала, т.е. вам будет известен URI содержимого этого сайта и вы сможете использовать его в своих медиаклиентах.

Организация доступа к этому содержимому из Silverlight предельно проста: либо в элементе *Source MediaElement* задается местоположение содержимого, которое будет воспроизводиться, как было показано ранее, либо источник (*Source*) задается в коде объектом URI.

Рассмотрим пример реализации этого на C#.

Сначала определяем в XAML элемент управления *MediaElement*:

```
<MediaElement x:Name="mel"></MediaElement>
```

Затем задаем в коде источник, используя URI:

```
Uri theUri = new
    Uri("http://web.sldrm.video.msn.com/d1/drm/ElephantsDream_WVC1_WMA9Pro_AES.wmv");

mel.Source = theUri;
mel.Play();
```

Такую простоту обеспечивает тот факт, что содержимое, зашифрованное с помощью средства разработки ПО (SDK) PlayReady, включает в заголовках адрес сервера лицензирования. В данном случае, сервер лицензирования находится по адресу *http://playready.directtaps.net/sl/rightsmanager.asmx*. Если вы проследите свою сесию, то увидите что Silverlight подключается к содержимому, считывает заголовки и затем после индивидуализации делает попытку обратиться к диспетчеру прав.

Если содержимое защищено с помощью WMDRM, а не PlayReady, такого заголовка нет. Поэтому *MediaElement* необходимо будет сообщить адрес сервера лицензирования. Для этого используется свойство *LicenseServerUriOverride* (Переопределение адреса сервера лицензирования) объекта *LicenseAcquirer* (Получатель лицензии) *MediaElement*.

Рассмотрим пример:

```
mel.LicenseAcquirer.LicenseServerUriOverride = new Uri("http://playready.directtaps.net/sl/rightsmanager.asmx",
UriKind.Absolute);
```

Определение текущего состояния

Свойство *CurrentState* (Текущее состояние) *MediaElement* может использоваться для определения текущего состояния в рассматриваемом выше процессе. *CurrentState* имеет два возможных значения, определенных перечислением *MediaElementState* (Состояние медиаэлемента), *Individualizing* или *AcquiringLicense*, которые представляют шаги 3 и 4 на рис. 10-16.

Таким образом, обработчик события *CurrentStateChanged* элемента *MediaElement* может быть настроен следующим образом:

```
<MediaElement x:Name="mel" CurrentStateChanged="mel_CurrentStateChanged"></MediaElement>
```

И после этого можно информировать пользователя о текущем состоянии следующим образом:

```
private void mel_CurrentStateChanged(object sender, RoutedEventArgs e)
{
   if (mel.CurrentState == MediaElementState.AcquiringLicense)
   {
      // Сообщаем пользователю, что Silverlight запрашивает лицензию для DRM
   }
   else if(mel.CurrentState == MediaElementState.Individualizing)
   {
      // Сообщаем пользователю, что идет процесс индивидуализации
   }
}
```

MediaElementState также включает соответствующие состояния для буферизации и воспроизведения мультимедиа. Это было рассмотрено в данной главе ранее.

Отказ пользователя от DRM и обработка ошибок DRM

Пользователь может отказаться от DRM, запретив воспроизведение защищенного содержимого. Для этого необходимо щелкнуть правой кнопкой мыши Silverlight-содержимое и выбрать опцию Configure Silverlight (Настроить Silverlight).

На рис. 10-18 представлена вкладка Playback (Воспроизведение) диалогового окна Microsoft Silverlight Configuration (Настройка Microsoft Silverlight).

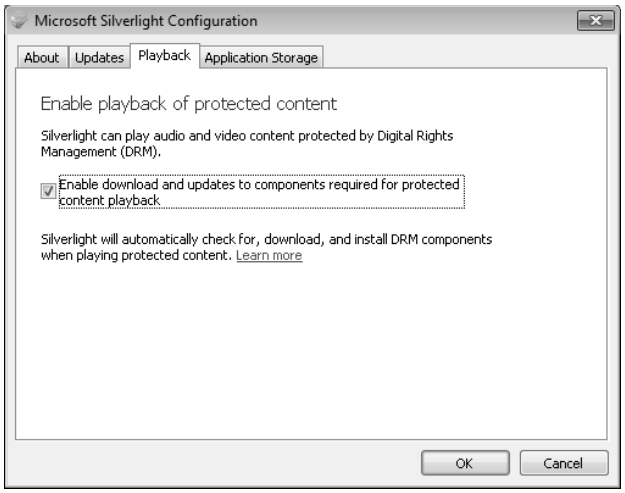

РИС. 10-18 Диалоговое окно настройки DRM для Silverlight.

Опция Enable Download And Updates (Включить загрузку и обновления) на вкладке Playback обеспечивает включение и выключение запроса на индивидуализацию. Если пользователь отменяет выбор этой опции, попытка доступа к защищенному содержимому приведет к тому, что медиаэлемент Silverlight сформирует новое событие *MediaFailed* (Сбой мультимедиа) с кодом ошибки 6008 и описанием DRM_E_INDIVIDUALIZATION_DISALLOWED (индивидуализация запрещена).

Остальные возможные ошибки представлены в табл. 10-1.

Таблица 10-1 Возможные ошибки при доступе к защищенным DRM медиаданным

Код ошибки / описание	Причины
6000 DRM_E_UNABLE_TO_PLAY_PROTECTED_CONTENT	*MediaElement* не может воспроизвести содержимое.
6001 DRM_E_INDIV_DOWNLOAD_FAILURE	Не удалось загрузить и выполнить ПО, полученное от сервера индивидуализации, и, таким образом, индивидуализация дала сбой.
6002 DRM_E_LICENSE_ACQUISITION_FAILURE	Сбой запроса лицензии или Silverlight не может выполнить доступ к серверу лицензирования.
6003 DRM_E_INDIV_INSTALL FAILURE	Не удалось установить компоненты индивидуализации.
6004 DRM_E_SILVERLIGHT_CLIENT_REVOKED	Срок действия компонентов индивидуализации истек, и они требуют обновления.
6005 DRM_E_INVALID_PROTECTED_FILE_HEADER	Сбой при обработке заголовка файла. Этот заголовок включает информацию DRM, поэтому невозможно продолжать работу. Скорее всего, заголовок неправильно сформирован.
6006 DRM_E_LICENSE_PROCESSING_FAILURE	Сбой при обработке лицензии. Вероятно, проблемы при загрузке лицензии.
6007 DRM_E_LICENSE_ACQUISITION_SERVICE_SPECIFIC	PlayReady допускает использование

	пользовательских ошибок. Эта ошибка используется именно для этого.
6008 DRM_E_INDIVIDUALIZATION_DISALLOWED	Пользователь отключил DRM в настройках конфигурации, как обсуждалось ранее.

Заключение

Данная глава представляет экскурс по воспроизведению мультимедиа в Silverlight с помощью элемента управления *MediaElement*. Мы рассмотрели его программирование с использованием API .NET и JavaScript и увидели, как задается источник мультимедиа и выполняется управление воспроизведением этого источника. Также мы научились управлять состояниями *MediaElement* и разобрались с буферизацией и уведомлением об этом пользователей. Были продемонстрированы возможности использования в Silverlight нового видеокодека H.264 и дан обзор архитектуры DRM в Silverlight и использования Silverlight для воспроизведения защищенного видео.

Глава 11

Мультимедиа в Silverlight: создание насыщенных изображений

Основным коммерческим сценарием в Веб сегодня является использование изображений и создание насыщенных изображений для обеспечения высококлассных сценариев взаимодействия с пользователем. Это та область, в которой чувствуется особая мощь Microsoft Silverlight благодаря предоставляемым им технологиям, таким как Deep Zoom и Microsoft Live Labs Photosynth. Данная глава посвящена этим технологиям и их применению в приложениях. Базовый элемент управления *Image*, обеспечивающий базовую поддержку обработки изображений, рассматривается в Главе 5, «XAML: трансформация и анимация».

Реализация Deep Zoom с помощью элемента управления *MultiScaleImage*

Deep Zoom обеспечивает новый уникальный способ управления изображениями в приложениях. Эта технология реализована в Extensible Application Markup Language (XAML) посредством элемента *MultiScaleImage* (Масштабируемое изображение), который, как следует из его имени, предоставляет возможность управлять размером и масштабом изображений, а Silverlight, в свою очередь, обеспечивает огромное виртуальное пространство, в котором эти изображения могут быть отрисованы.

Deep Zoom позволяет создавать изображения с высочайшим разрешением и отображать их без предварительной полной загрузки. Рассмотрим такой сценарий: имеется очень детализированное изображение размером 2 гигабайта и с разрешением, во много раз превышающим разрешение экрана. Как правило, для отображения такого изображения создается его Веб-версия с меньшим разрешением и меньшим размером файла, что позволяет Веб-пользователям загружать и просматривать его. Конечно, это приводит к существенной потере качества изображения.

А теперь обратимся к онлайн картам. По сути, карта – это огромное изображение, разделенное на фрагменты, которые можно объединять для просмотра всей области. Когда требуется просмотреть определенную часть карты, картографическое программное обеспечение загружает и отображает соответствующий фрагмент. Это же программное обеспечение предоставляет фрагменты, необходимые для реализации изменения масштаба карты. Deep Zoom имеет аналогичный принцип работы. В обсуждаемом выше сценарии с двухгигабайтным изображением его можно разбить на фрагменты, отображаемые при разном масштабе. Первой пользователю предлагается версия изображения с низким разрешением, т.е. в состоянии «без увеличения». Когда пользователь увеличивает масштаб, Silverlight загружает соответствующие фрагменты для формирования визуального представления изображения с заданным масштабом. Таким образом, можно полностью сохранить точность изображения, но при этом будут загружаться только те его фрагменты, которые просматриваются пользователем в данный момент.

Кроме того, технология Deep Zoom позволяет встраивать изображение с одним масштабом в изображение с другой степенью увеличения, что открывает возможности для создания множества различных спецэффектов. Лучше всего рассмотреть это на примере. На рис. 11-1 представлена иллюстрация к детскому научному проекту.

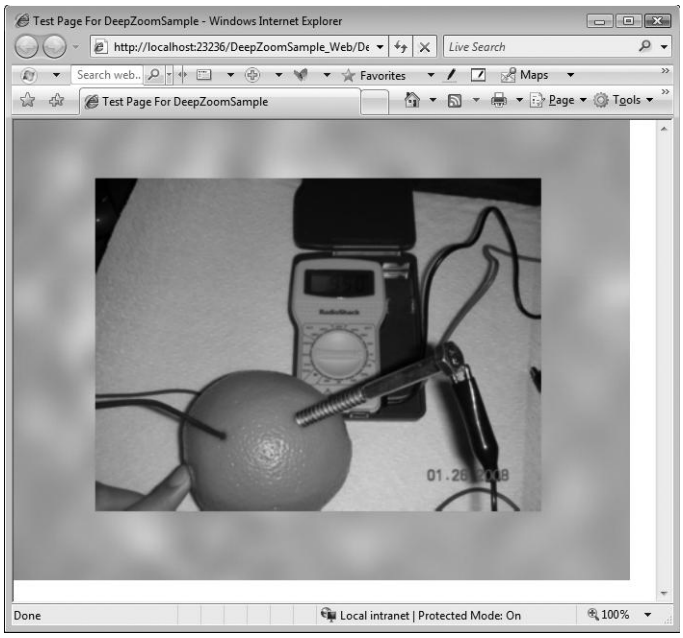

РИС. 11-1 Изображение в Deep Zoom.

Пока что ничего сверхъестественного, казалось бы, обычная фотография. Так чем же вызван весь этот шум вокруг Deep Zoom? В этом приложении можно менять масштаб, прокручивая колесо мыши. Я уменьшил масштаб, чтобы показать весь рисунок, и теперь на рис. 11-2 можно увидеть первое изображение относительно всей фотографии. Изображение с рис. 11-1 выглядит, как наклейка на апельсине, но если увеличить масштаб, его можно увидеть абсолютно четко.

Посмотрите внимательно на апельсин на рис. 11-2. Как видите, у него в центре располагается изображение с рис. 11-1. Увеличьте масштаб, и вы получите изображение, представленное на рис. 11-3.

Таким образом, все второе изображение с рис. 11-2 – не более чем пиксел в зрачке глаза на изображении рис. 11-3, а размер первого изображения с рис. 11-1 всего несколько пикселов на втором рисунке. По этим нескольким статическим рисункам сложно оценить возможность Deep Zoom должным образом. Deep Zoom, действительно, надо увидеть, чтобы поверить и понять, почему эта технология называется Deep Zoom: она позволяет организовывать изображения так, что вы можете увеличивать и уменьшать масштаб изображений, размещая их на гигантских холстах.

РИС. 11-2 Исходное изображение в другом масштабе.

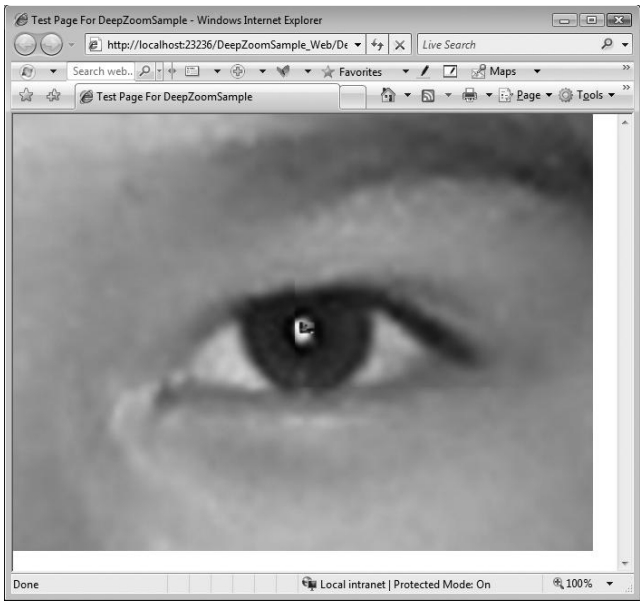

РИС. 11-3 Еще более глубокое масштабирование.

Использование редактора Deep Zoom Composer

Базовое приложение, которое позволит уменьшать и увеличивать масштаб описанным выше образом, написать довольно просто. Для этого возьмем элемент управления *MultiScaleImage* и ассоциируем его с файлом, содержащим метаданные изображений. Файл метаданных создается в редакторе Deep Zoom Composer, который можно загрузить в Центре загрузки Microsoft. Deep Zoom Composer представлен на рис. 11-4.

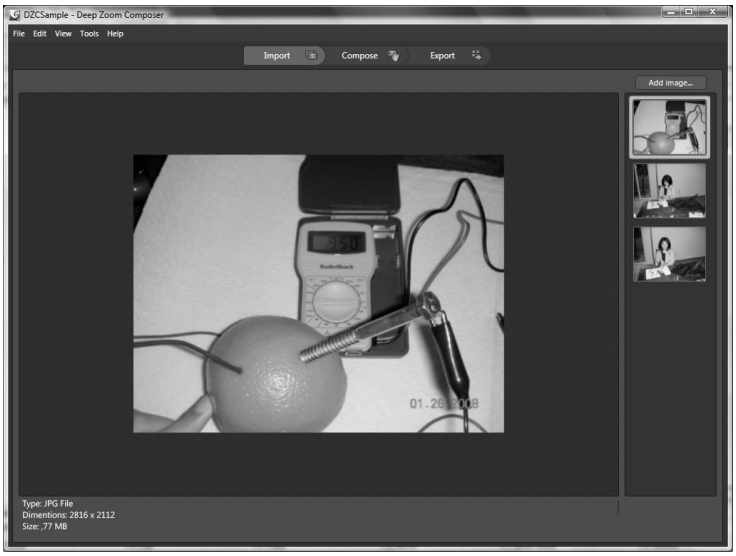

РИС. 11-4 Редактор Deep Zoom Composer.

Deep Zoom Composer выполняет простую последовательность операций, включающую *Import* (Импорт), за которым следует *Compose* (Компоновка) с последующим *Export* (Экспорт). Итак, начинаем с вкладки Import, щелкаем Add Image (Добавить изображение), чтобы выбрать изображение, и повторяем этот шаг для всех изображений, которые хотим использовать. На рис. 11-4 можно увидеть, что добавлены три изображения.

Следующий шаг – компоновка, которая осуществляется с помощью опций на вкладке Compose. На этой вкладке изображения размещаются на соответствующих поверхностях. Вы можете менять масштаб изображений, чтобы разместить в них другие изображения. Например, на рис. 11-5 можно увидеть, что я взял одно из изображений и увеличил его масштаб так, что на экране виден только глаз.

Новые изображения размещаются в своем стандартном разрешении на выбранных уровнях увеличения. На рис. 11-5 можно видеть, куда вставлено второе изображение (в зрачок глаза). Позже, при выполнении приложения, увидеть это изображение можно будет, только увеличив первое. На рис. 11-6 показано, куда помещается второе изображение на первом.

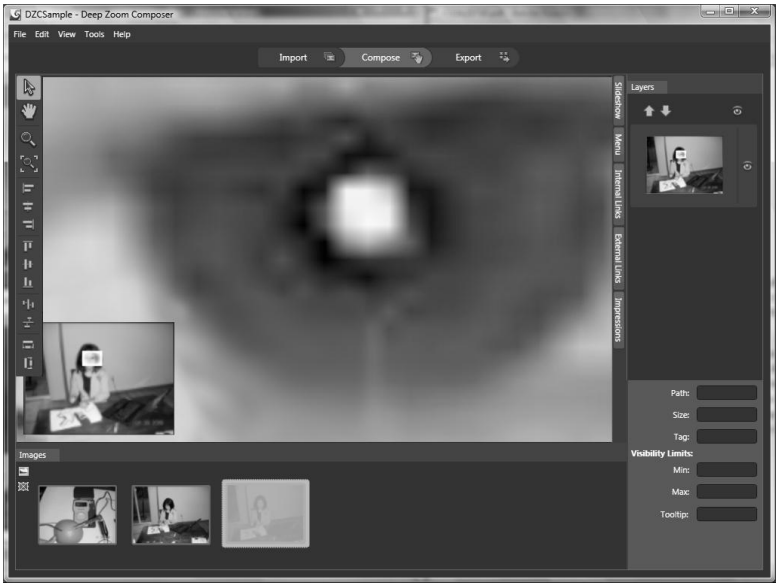

РИС. 11-5 Компоновка изображений для просмотра с разными уровнями увеличения в редакторе Deep Zoom Composer.

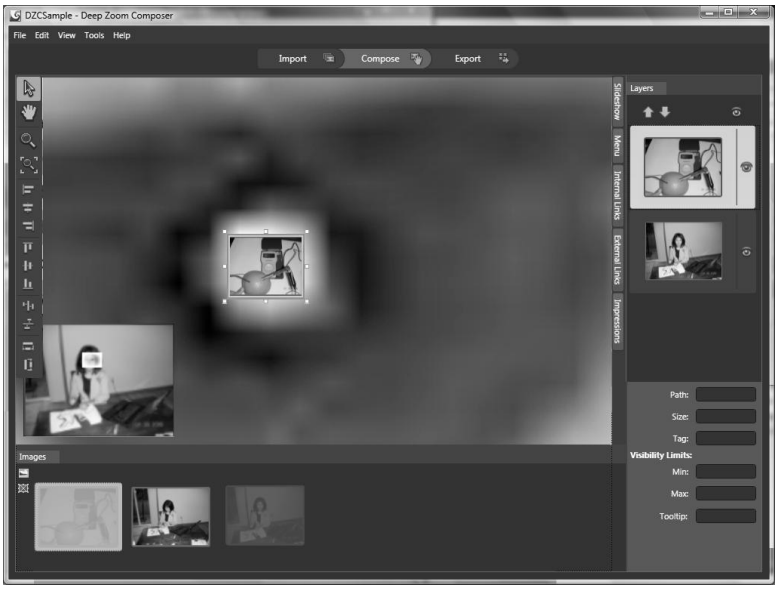

РИС. 11-6 Добавление нового изображения на увеличенное первое.

Этот простой пример демонстрирует, как можно добавлять в изображение другое изображение, которое можно увидеть, только увеличив первое. Deep Zoom позволяет создавать намного боле сложные приложения, данный пример лишь дает представление о возможностях этой технологии.

Теперь мы готовы перейти к третьему этапу – экспорту. Для этого будем использовать вкладку Custom вкладки Export редактора Deep Zoom Composer (рис. 11-7).

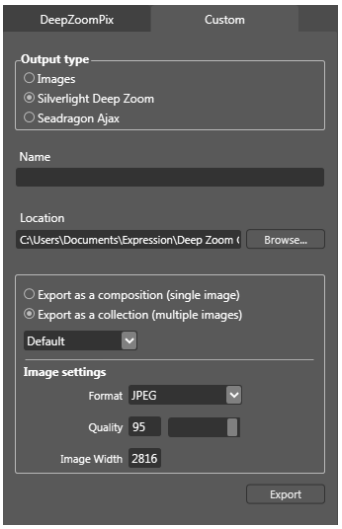

РИС. 11-7 Экспорт метаданных Deep Zoom.

На вкладке Export имеется несколько опций. Предлагается возможность выполнить экспорт в сервис DeepZoomPix, бесплатный сервис на базе Deep Zoom для публикации фотографий, или в готовое Silverlight-приложение, которое можно использовать в своих решениях. Для этого убедитесь, что выбрана вкладка Custom (как на рис. 11-7) и выберите проект Silverlight Deep Zoom.

Чтобы экспортировать метаданные изображений, скомпонованных в Deep Zoom, необходимо присвоить имя проекту и экспортировать проект в заданный каталог. При этом в выходном каталоге Вы найдете папку GeneratedImages. Обратите внимание на файл SparseImageSceneGraph.xml, который является конфигурационным, который определяет каждое изображение и его местоположение на других изображениях с разными уровнями увеличения. Например, рассмотрим граф сцены для XAML с двумя изображениями:

```
<?xml version="1.0"?>
<SceneGraph version="1">
  <AspectRatio>1.33333333333334</AspectRatio>
  <SceneNode>
  <FileName>C:\Code\SLBook\Chapter11\DZCSample
     \source images\dzcsample\DSCN2961.JPG</FileName>
    <x>0</x>
    <y>0</y>
    <Width>1</Width>
    <Height>1</Height>
    <ZOrder>1</ZOrder>
  </SceneNode>
  <SceneNode>
  <FileName>C:\Code\SLBook\Chapter11\DZCSample
     \source images\dzcsample\DSCN2959.JPG</FileName>
    <x>0.451782754964542</x>
    <y>0.313488814592021</y>
    <Width>0.00099432659277551</Width>
    <Height>0.00099432659277551</Height>
    <ZOrder>2</ZOrder>
  </SceneNode>
</SceneGraph>
```

Как видите, это довольно простой XML-код. В нем указывается соотношение сторон основного изображения (полученное, исходя из размеров первого изображения), и каждое изображение превращается в *SceneNode* (Узел сцены). Первое изображения является первым *SceneNode*. Оно располагается в позиции (0,0) и является нормализованным изображением, т.е. его ширина и высота принимаются за 1. Размеры и расположение всех остальных изображений устанавливаются относительного первого.

Второе изображение размещается в точке с координатами 0,45 по оси x и 0,31 по оси y. Его размер составляет, примерно, 0,00099 размеров первого изображения. Таким образом, если увеличить первое изображение, примерно, в 10000 раз, можно будет увидеть второе изображение. Его Z-порядок равен 2 (Z-порядок первого изображения – 1), т.е. он будет отрисовываться поверх первого изображения.

Кроме того, редактор Deep Zoom Composer разделяет изображение на равные фрагменты, поэтому при масштабировании рисунка загружается не все изображение, а лишь просматриваемые фрагменты, что очень эффективно при работе с большими изображениями. При просмотре изображения в маленьком масштабе, вы видите все фрагменты уменьшенными, следовательно, с меньшим разрешением. При увеличении масштаба изображения до пределов разрешающей способности (или выше), вы будете видеть лишь часть рисунка, т.е. потребуется загружать только фрагменты, представляющие видимую часть изображения, что обеспечивает экономное использование пропускного канала и сокращение времени загрузки.

Остальные экспортируемые файлы представляют собой набор XML-файлов, содержащих все сведения о фрагментах и их положении относительно основного изображения. Они располагаются в каталоге, созданном редактором Deep ZoomComposer, в котором вы также найдете ряд подпапок с изображениями. Поскольку это XML-файлы, их можно открыть и просмотреть.

Первый проект Deep Zoom

Один из вариантов экспорта из Deep Zoom Composer – Silverlight Deep Zoom. Обычно это приложение располагается в каталоге C:\Users\<*вашеимя*>\Documents\Expression\Deep Zoom Composer Projects\<*названиевашегопроекта*>\Exported Data.

Deep Zoom Composer после экспорта в Silverlight Deep Zoom автоматически дополнительно предлагает просмотреть работу в браузере.

В созданном приложении Silverlight встроена функциональность управления мышью. Благодаря этому вы можете перемещаться по изображению или менять его масштаб. Как это сделать, будет рассмотрено в следующем разделе.

На основе сгенерированных файлов Вы можете самостоятельно сделать свой первый проект Deep Zoom. Создайте проект Silverlight приложения в Microsoft Visual Studio. В папку *ClientBin* проекта веб-приложения ASP.NET скопируйте и включите полученную при экспорте папку *GeneratedImages*.

Формирование представления содержимого Deep Zoom осуществляется с помощью элемента управления *MultiScaleImage*. Добавьте *MultiScaleImage* в файл MainPage.xaml, в качестве значения его свойства *Source* задайте местоположение XML-файла с метаданными скомпонованного изображения. В данном случае имя файла будет примерно таким: /GeneratedImages/dzc_output.xml. Используя другие атрибуты *MultiScaleImage*, можно настроить визуальное представление страницы, например, ее высоту и ширину.

```
<MultiScaleImage x:Name="msi" Source="/GeneratedImages/dzc_output.xml"/>
```

При выполнении этого приложения элемент управления *MultiScaleImage* формирует визуальное представление верхнего элемента скомпонованного изображения.

Использование мыши и логических координат в Deep Zoom

Как и все остальные элементы управления Silverlight, для *MultiScaleImage* можно определить методы обработки событий. Например, для перемещения изображения используются обычные события мыши: *MouseLeftButtonDown, MouseLeftButtonUp* и *MouseMove*, – и обрабатываются они так же, как это делается при перемещении любого элемента методом «drag-and-drop».

Сначала рассмотрим XAML для *MultiScaleImage*:

```
<MultiScaleImage x:Name="msi" Source="/GeneratedImages/dzc_output.xml"/>
```

Теперь более подробно остановимся на функциональности. Прежде всего, добавим код, используемый всем приложением, который обеспечивает отслеживание текущего состояния мыши и просматриваемых в настоящий момент координат *MultiScaleImage*:

```
double zoom = 1;
bool duringDrag = false;
bool mouseDown = false;
Point lastMouseDownPos = new Point();
Point lastMousePos = new Point();
Point lastMouseViewPort = new Point();
```

Вот что происходит, когда пользователь щелкает изображение:

```
this.MouseLeftButtonDown += delegate(object sender, MouseButtonEventArgs e)
{
   lastMouseDownPos = e.GetPosition(msi);
   lastMouseViewPort = msi.ViewportOrigin;

   mouseDown = true;

   msi.CaptureMouse();
};
```

В этом примере предполагается, что пользователь двигает мышью, удерживая ее кнопку, поэтому задаем булевому свойству *mouseDown* значение *true*. Также мы хотим знать местоположение указателя мыши, которое можно сохранить в объекте *Point*, и исходное положение мыши, которое также может храниться в *Point*. Наконец, требуется, чтобы *MultiScaleImage* перехватывал все команды мыши, сделать это позволит метод *CaptureMouse*.

Обычно, если кнопка мыши удерживается, значит, пользователь начинает перетягивание. При этом мы хотим обеспечить ему возможность перемещения по изображению. Вот код, реализующий это:

```
this.MouseMove += delegate(object sender, MouseEventArgs e)
{
 lastMousePos = e.GetPosition(msi);

 if (duringDrag)
 {
  Point newPoint = lastMouseViewPort;
  newPoint.X += (lastMouseDownPos.X - lastMousePos.X) /
              msi.ActualWidth * msi.ViewportWidth;
  newPoint.Y += (lastMouseDownPos.Y - lastMousePos.Y) /
              msi.ActualWidth * msi.ViewportWidth;
  msi.ViewportOrigin = newPoint;
 }
};
```

Событие *MouseMove* формируется независимо от того, удерживается кнопка мыши или нет, поэтому для обозначения перетягивания используем переменную *duringDrag*. Если пользователь выполняет перетягивание, выполняется код выражения *if{…}*.

В данном случае, чтобы определить новые координаты *X*,*Y* указателя мыши в *MultiScaleImage*, изменение каждой координаты делим на произведение фактической ширины *MultiScaleImage* и ширины окна просмотра, т.е. на логическую ширину. На забывайте, что *MultiScaleImage* позволяет как увеличивать, так и уменьшать масштаб, поэтому местоположение в окне просмотра не всегда совпадает с положением на экране, в котором оно находится. Данный код обеспечивает вычисление местоположения. Далее заменяем координаты текущего верхнего левого угла окна просмотра координатами нового положения указателя мыши. Это обеспечивает эффект перетягивания изображения.

Наконец, когда пользователь отпускает кнопку мыши, необходимо выполнить некоторые вспомогательные операции. Прежде всего, если не поддерживается функциональность колеса мыши, требуется проверить, не нажимал ли пользователь клавишу Shift для перехода в полноэкранный режим. Кроме того, не забудьте очистить булевы функции, сохраняющие состояния удержания кнопки мыши и перетягивания.

```
this.MouseLeftButtonUp += delegate(object sender, MouseButtonEventArgs e)
{
 if (!duringDrag)
 {
  bool shiftDown = (Keyboard.Modifiers & ModifierKeys.Shift) == ModifierKeys.Shift;
  double newzoom = zoom;

  if (shiftDown)
  {
   newzoom /= 2;
  }
  else
  {
   newzoom *= 2;
  }

  Zoom(newzoom, msi.ElementToLogicalPoint(this.lastMousePos));
 }
 duringDrag = false;
 mouseDown = false;

 msi.ReleaseMouseCapture();
};
```

Эти три простые метода будут обеспечивать пользователю возможность перетягивать изображение и перемещаться по нему независимо от степени его увеличения. Также нужно описать вспомогательный метод *Zoom*, используемый в предыдущем фрагменте кода:

```
  private void Zoom(double newzoom, Point p)
     {
     if (newzoom < 0.5)
     {
     newzoom = 0.5;
     }
     msi.ZoomAboutLogicalPoint(newzoom / zoom, p.X, p.Y);
     zoom = newzoom;
     }
```

В следующем разделе будет показано, как с помощью колеса мыши увеличивать и уменьшать изображение и, таким образом, открывать части изображения, скрытые при других степенях увеличения.

Реализация масштабирования с использованием колеса мыши

Проблемой при создании приложений Deep Zoom является то, что стандартным средством изменения масштаба изображения, де факто, является колесо мыши, но Silverlight и .NET Framework не обрабатывают события колеса мыши. Существует два варианта решения этой проблемы. Первый – использовать JavaScript вместо C#, потому что браузер может перехватывать перемещения колеса мыши и формировать события в ответ на его прокручивание. Второй вариант – использовать механизмы Silverlight по взаимодействию с браузером, т.е. браузер будет перехватывать событие и затем информировать .NET об этом. Вся дальнейшая обработка будет осуществляться в .NET. Все намного проще, чем кажется[1].

[1] В версиях до Deep Zoom Composer Febrary 2009 Release при экспорте генерировался также Silverlight проект для Visual Studio со всем необходимым кодом. (*прим. редактора*)

Добавим в Silverlight проект класс *MouseWheelHelper.cs*. Включите в этот файл следующее пространство имен:

```
using System.Windows.Browser;
```

В файл *MouseWheelHelper.cs* допишем следующий код:

```
public class MouseWheelEventArgs : EventArgs
 {
  private double delta;
  private bool handled = false;

  public MouseWheelEventArgs(double delta)
  {
   this.delta = delta;
  }

  public double Delta
  {
   get { return this.delta; }
  }

  // Используйте обработчик для перехвата стандартного поведения в браузере!
  public bool Handled
  {
   get { return this.handled; }
   set { this.handled = value; }
  }
 }

 public class MouseWheelHelper
 {

  public event EventHandler<MouseWheelEventArgs> Moved;
  private static Worker worker;
  private bool isMouseOver = false;

  public MouseWheelHelper(FrameworkElement element)
  {

   if (MouseWheelHelper.worker == null)
    MouseWheelHelper.worker = new Worker();

   MouseWheelHelper.worker.Moved += this.HandleMouseWheel;

   element.MouseEnter += this.HandleMouseEnter;
   element.MouseLeave += this.HandleMouseLeave;
   element.MouseMove += this.HandleMouseMove;
  }

  private void HandleMouseWheel(object sender, MouseWheelEventArgs args)
  {
   if (this.isMouseOver)
    this.Moved(this, args);
  }

  private void HandleMouseEnter(object sender, EventArgs e)
  {
   this.isMouseOver = true;
  }

  private void HandleMouseLeave(object sender, EventArgs e)
  {
   this.isMouseOver = false;
  }

  private void HandleMouseMove(object sender, EventArgs e)
  {
   this.isMouseOver = true;
  }

  private class Worker
```

```
{
  public event EventHandler<MouseWheelEventArgs> Moved;

  public Worker()
  {
    if (HtmlPage.IsEnabled)
    {
      HtmlPage.Window.AttachEvent("DOMMouseScroll", this.HandleMouseWheel);
      HtmlPage.Window.AttachEvent("onmousewheel", this.HandleMouseWheel);
      HtmlPage.Document.AttachEvent("onmousewheel", this.HandleMouseWheel);
    }

  }

  private void HandleMouseWheel(object sender, HtmlEventArgs args)

  {
    double delta = 0;

    ScriptObject eventObj = args.EventObject;

    if (eventObj.GetProperty("wheelDelta") != null)
    {
      delta = ((double)eventObj.GetProperty("wheelDelta")) / 120;


      if (HtmlPage.Window.GetProperty("opera") != null)
        delta = -delta;
    }
    else if (eventObj.GetProperty("detail") != null)
    {
      delta = -((double)eventObj.GetProperty("detail")) / 3;

      if (HtmlPage.BrowserInformation.UserAgent.IndexOf("Macintosh") != -1)
        delta = delta * 3;
    }

    if (delta != 0 && this.Moved != null)
    {
      MouseWheelEventArgs wheelArgs = new MouseWheelEventArgs(delta);
      this.Moved(this, wheelArgs);

      if (wheelArgs.Handled)
        args.PreventDefault();
    }
  }
 }
}
```

Следующий код:

```
HtmlPage.Window.AttachEvent("DOMMouseScroll", this.HandleMouseWheel);
HtmlPage.Window.AttachEvent("onmousewheel", this.HandleMouseWheel);
HtmlPage.Document.AttachEvent("onmousewheel", this.HandleMouseWheel);
```

использует механизм доступа к объектной модели браузера из Silverlight для связывания JavaScript-событий и .NET-событий. (Подробно механизм доступа к браузеру обсуждается в Главе 7, «Механизм доступа к объектной модели браузера».) В ответ на возникновение любого из заданных событий JavaScript формируется .NET-событие *HandleMouseWheel* (Обработчик событий колеса мыши).

Метод *HandleMouseWheel* принимает в качестве аргумента объект *HtmlEventArgs*. Одним из дочерних объектов *HtmlEventArgs* является объект *EventObject* (Событие). *EventObject* содержит метаданные колеса мыши, такие как *wheelDelta* (Перемещение колеса), которое обозначает изменение направления в результате прокручивания колеса мыши. Разные браузеры, например Opera, возвращают разные значения, поэтому здесь используется код для обработки этой проблемы.

Ну и, наконец, в файле MainPage.xaml.cs необходимо дописать следующее:

```
new MouseWheelHelper(this).Moved += delegate(object sender, MouseWheelEventArgs e)
    {
     e.Handled = true;

     double newzoom = zoom;

     if (e.Delta < 0)
       newzoom /= 1.3;
     else
       newzoom *= 1.3;

     Zoom(newzoom, msi.ElementToLogicalPoint(this.lastMousePos));
     msi.CaptureMouse();
    };
```

Коллекции в Deep Zoom

Кроме возможности увеличения и уменьшения изображения, можно также создавать *коллекции* изображений и работать с ними в масштабируемой среде.

Возможно, вы заметили, что при экспорте изображений из Deep Zoom Composer предлагается опция Create Collection (Создать коллекцию). Как следует из ее имени, эта опция обеспечивает создание коллекции изображений. На рис. 11-8 мы видим в рабочей области множество изображений, их можно экспортировать как коллекцию.

При экспорте изображений как коллекции, кроме файла dzc_output.xml, для каждого изображения создается XML-файл и файл SparseImageSceneGraph.xml, в котором сохраняются данные о размещении каждого изображения.

Для работы с изображениями коллекции Silverlight предлагает коллекцию *SubImages* (Множество изображений). Обратите внимание, что при формировании визуального представления страницы полный набор изображений не загружается, поэтому коллекция будет пустой в этот момент. Чтобы иметь возможность работать с коллекцией, необходимо обработать событие *ImageOpenSucceeded* (Изображение открыто успешно). Это событие формируется в случае успешной загрузки и отображения.

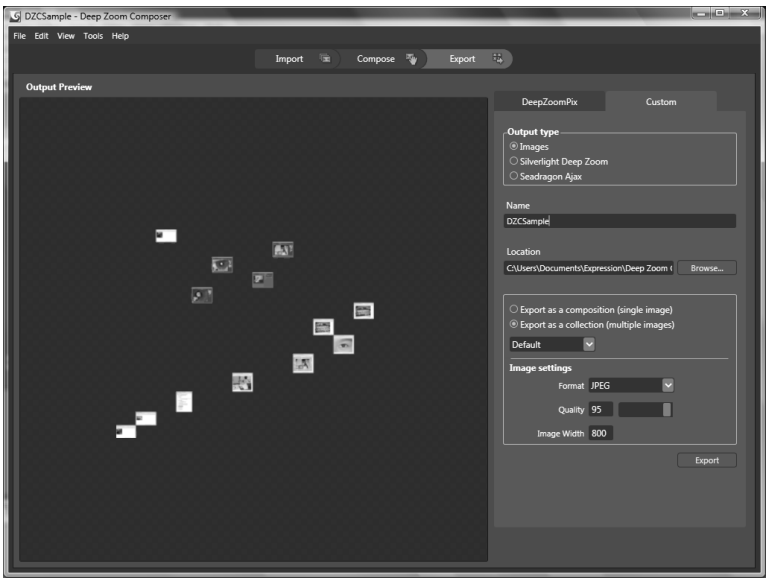

РИС. 11-8 Создание коллекции Deep Zoom.

Рассмотрим XAML:

```
<MultiScaleImage x:Name="dz"
  Source="/GeneratedImages/dzc_output.xml"
  ImageOpenSucceeded="dz_ImageOpenSucceeded"
  Height="480" Width="640">
</MultiScaleImage>
```

В этот момент коллекция *SubImages* будет заполнена и может использоваться для работы с отдельными рисунками, ее составляющими. Рассмотрим пример изменения месторасположения изображений:

```
private void dz_ImageOpenSucceeded(object sender, RoutedEventArgs e)
{
  int nImages = dz.SubImages.Count;
  for (int lp = 0; lp < nImages; lp++)
  {
    dz.SubImages[lp].ViewportOrigin = new Point(lp*0.1, lp*0.1);
  }
}
```

Как видите, процесс абсолютно аналогичен тому, который использовался ранее для работы с одним рисунком. В случае с коллекцией, работа с каждым изображением осуществляется через *SubImages[lp]*, где *lp* – переменная цикла, которая может принимать значения от 0 до числа, соответствующего количеству изображений в коллекции. Результаты представлены на рис. 11-9. Обратите внимание на разницу между этой и исходной компоновкой, представленной на рис. 11-8.

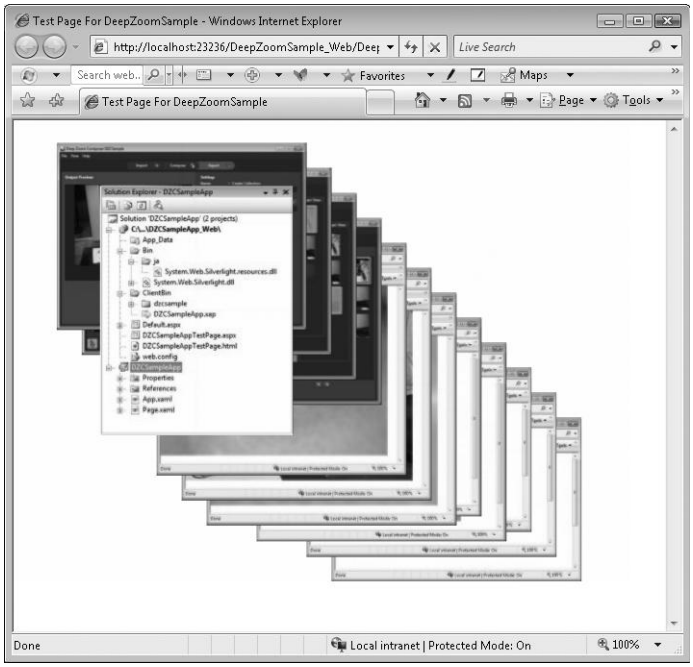

РИС. 11-9 Компоновка коллекции изображений.

Deep Zoom Composer позволяет также выполнять наложение изображений, делая их масштабируемыми. Вы можете не иметь фотоаппарата, который позволяет создавать двухгигабайтные изображения с высочайшим разрешением, но можете сделать множество кадров одной сцены и наложить их друг на друга, чтобы обеспечить аналогичный эффект.

Скажем, имеется 4-мегапиксельный фотоаппарат. С его помощью можно сделать только 4-мегапиксельные снимки, которые при большом увеличении не будут настолько четкими, как (например) снимки, сделанные 400-мегапиксельным фотоаппаратом.

Но если сделать множество снимков интересующего объекта с большим приближением и затем наложить их друг на друга, можно скомпоновать изображение с большим разрешением, чем обеспечивает обычный снимок. Deep Zoom Composer позволяет делать именно это.

Например, на рис. 11-10 и 11-11 представлены два из пяти снимков длинного вертикального панно, которое я нашел на одном из Пекинских рынков.

РИС. 11-10 Верхняя часть картины.

РИС. 11-11 Снимок той же картины, но со смещением точки обзора несколько ниже.

Deep Zoom Composer позволяет импортировать такие изображения и накладывать их друг на друга. Сначала запустим Deep Zoom Composer и импортируем изображения.

Затем на вкладке Compose добавьте все изображения. Не беспокойтесь об их компоновке. Наконец, выберите все изображения, щелкните правой кнопкой мыши одно из них и нажмите Create Panoramic Photo (Создать панораму). Эта опция представлена на рис. 11-12.

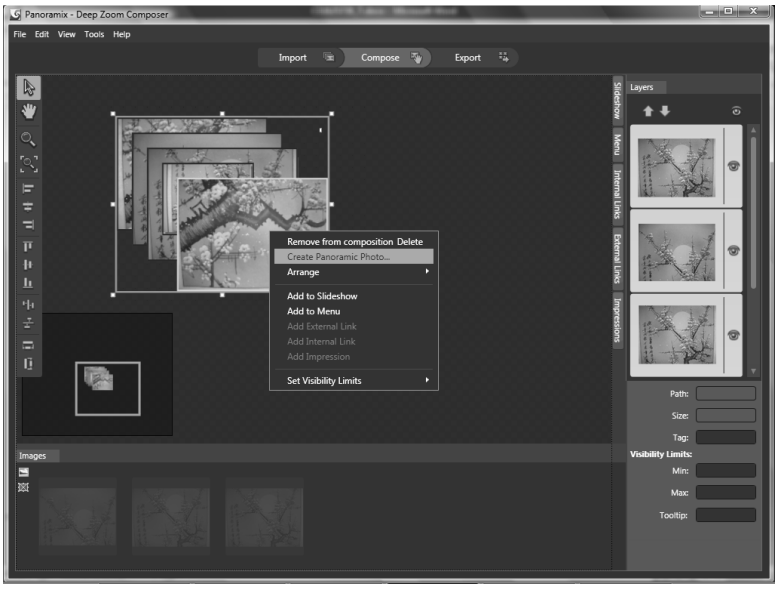

РИС. 11-12 Создание панорамного изображения в Deep Zoom Composer.

Deep Zoom Composer начинает наложение изображений и формирование составного изображения. Все происходит довольно быстро. Я выполнял этот пример на компьютере с Индексом производительности Windows равным 3.5, и весь процесс занял примерно 3 секунды.

В большинстве случаев могут возникнуть разногласия в выравнивании изображений. Deep Zoom Composer обеспечивает возможность предварительного просмотра панорамного изображения (Stitching Preview), как показано на рис. 11-13.

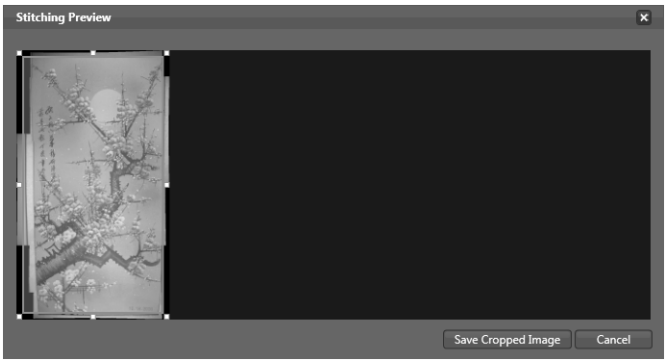

РИС. 11-13 Предварительный просмотр составного панорамного изображения.

В Stitching Preview изображение можно обрезать и сохранить. Deep Zoom Composer добавляет полученное изображение в область компоновки. После этого с ним можно работать, как с любым другим изображением.

Создание панорам с помощью Photosynth

Photosynth – технология, разработанная Microsoft Research и Microsoft Live Labs, которая обеспечивает создание сцены из существующих фотографических данных о ней. Например, вы сделали множество снимков одного объекта в разных ракурсах, но эти снимки все равно не отражают реальной картины, потому что, прежде всего, фотокамера может делать только прямоугольные снимки, снимок не может охватить всей сцены и не может обеспечить объемного изображения.

Photosynth позволяет через наложение большого числа плоских снимков создавать объемные изображения. Итак, представим, гуляя по городу, вы нашли живописный архитектурный ансамбль и сделали множество снимков. Затем, показывая эти фотографии друзьям, вы объясняете, что где находится, говоря, что вот это здание стоит слева от того, которое на другом снимке, и т.д. Photosynth делает это за вас. Он сопоставляет изображения, находит подобия и на основании этой информации

оценивает форму просматриваемого объекта в трехмерном пространстве, используя точку обзора каждой фотографии. Затем Photosynth воссоздает объект в трехмерном пространстве и размещает снимки в соответствующих местах.

Рис. 11-14 представляет пример синта изображений музея Лувр в Париже.

РИС. 11-14 Синт Парижа.

Загрузить редактор и программу просмотра Photosynth можно по адресу *http://photosynth.net/learn.aspx*. Приведенный здесь пример не обеспечивает по-настоящему реального представления. На рис. 11-14 центральный снимок ярче остальных. Передвигая изображение, можно создать иллюзию перемещения по трехмерному пространству, созданному на базе этих фотографий.

Photosynth также использует основополагающую технологию Deep Zoom: для просмотра части сцены, образованной несколькими сотнями изображений, не приходится загружать их все.

Создать собственный синт очень просто. Сначала необходимо сделать несколько фотографий интересующего объекта, немного изменяя ракурс. Например, встаньте в центре комнаты и сделайте первый снимок, затем немного повернитесь вправо и сделайте второй снимок и т.д.

На рис. 11-15 и 11-16 представлено два последовательных снимка интерьера.

РИС. 11-15 Один из снимков, составляющих синт.

РИС. 11-16 Другой снимок из того же синта.

Как видите, кресло присутствует на обоих снимках. Photosynth при создании синта сопоставляет эти снимки и замечает, что вторая картинка располагается правее правой. Для этого синта я сделал примерно 50 снимков комнаты.

Далее загружаем редактор Photosynth, который также позволяет просматривать синты в браузере. Пользователи других операционных систем, не Windows, могут использовать для просмотра синтов проигрыватель Silverlight. (Этот проигрыватель подробно рассматривается в следующем разделе.) Но создавать синты могут только пользователи Windows.

Photosynth можно запустить из меню Пуск или нажав кнопку Start A New Synth (Создать синт) на сайте Photosynth.net, как показано на рис. 11-17. Для входа понадобится Windows Live ID и этот же ID используется для загрузки и размещения создаваемых синтов.

РИС. 11-17 Запуск нового синта.

Появляется диалоговое окно Create Synth (Создать синт), как показано на рис. 11-18. Осталось лишь добавить фотоснимки и задать имя синта.

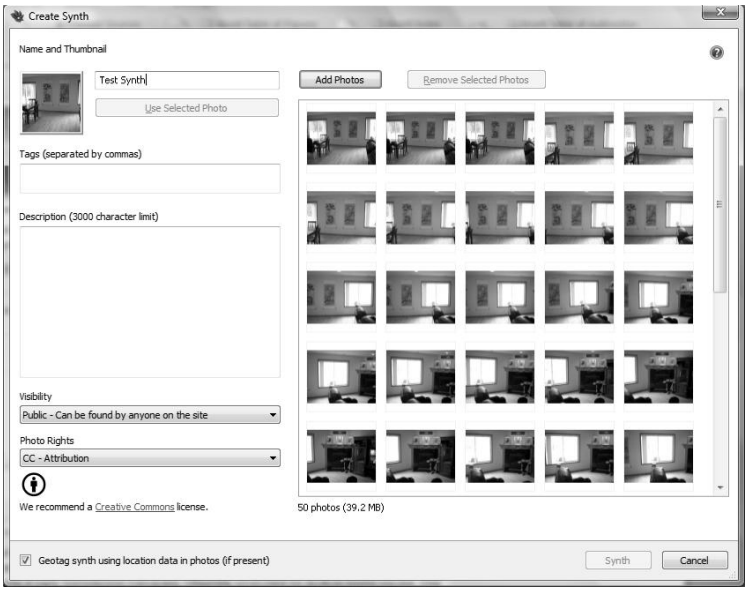

РИС. 11-18 Создание синта.

После этого остается щелкнуть кнопку Synth (Синтезировать), и Photosynth сделает все остальное сам. Готовый синт можно просмотреть в браузере (рис. 11-19). На этом рисунке показано, как изображения были синтезированы в одну трехмерную панораму. Обратите внимание на кривую внизу экрана. Она соответствует окружности, вдоль которой я разворачивался, когда делал снимки.

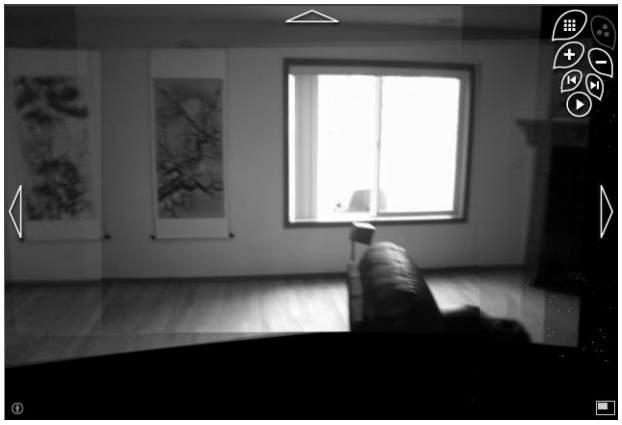

РИС. 11-19 Готовый синт.

Каждый синт имеет глобально уникальный идентификатор (globally unique identifier, GUID). URL созданного здесь синта:

http://photosynth.net/view.aspx?cid=98FF99B1-9B62-408A-9975-3A9F7E68E545

где *cid*= – это GUID. Он понадобится при просмотре синта в проигрывателе Silverlight. Если вы создадите собственный синт, вы получите для него другой GUID. Следует заметить, что в настоящее время все синты доступны для просмотра в двух режимах:

- Public (Общедоступный) – может быть просмотрен и найден на сайте Photosynth, кем угодно.
- Unlisted (Необъявленный) – может быть просмотрен только тем, кто, получил ссылку.

Использование Photosynth в Silverlight

Проигрыватель Photosynth на базе Silverlight доступен на сайте Photosynth.net. Чтобы просмотреть в нем синт, созданный в предыдущем разделе, просто введите такой URL:

http://photosynth.net/silverlight/photosynth.aspx?cid=98FF99B1-9B62-408A-9975-3A9F7E68E545

Для просмотра собственного синта подставьте его GUID в приведенный выше URL. Переход по этому URL обеспечивает загрузку XAP, содержащего приложение Photosynth, которое используется для загрузки синта. Как это выглядит на Mac в браузере Safari, показано на рис. 11-20.

РИС. 11-20 Использование Photosynth на Mac Silverlight.

В настоящее время просматривать собственные синты можно только на сайте Photosynth.net, но выход специального элемента управления не за горами. Подробнее смотрите на сайте Photosynth.

Заключение

В данной главе рассмотрены доступные в Silverlight технологии обработки изображений, включая Deep Zoom и Photosynth. Deep Zoom позволяет просматривать изображения чрезвычайно высокого разрешения в средах с ограниченной пропускной способностью путем загрузки фрагментов изображений для различных ракурсов и масштабов. Вы узнали, как использовать виртуальный холст для размещения одного изображения на разных уровнях увеличения другого, и как работать с коллекциями изображений. Представлен редактор Deep Zoom Composer, используемый для создания Silverlight-приложения, которое обеспечивает функциональность Deep Zoom, и показана возможность наложения нескольких менее информативных изображений для получения одного большого панорамного изображения.

Кроме того, все эти процессы обработки изображений были перенесены в трехмерное пространство с помощью Photosynth. Технология Photosynth позволяет создавать трехмерные изображения из плоских, двухмерных, путем сопоставления их содержимого и интерполяции угла обзора. Photosynth обладает ошеломляющими возможностями. Также мы показали, как благодаря Silverlight средство просмотра Photosynth может использоваться на разных платформах.

Глава 12

Создание Silverlight-приложений, взаимодействующих с сервером

В данной главе обсуждается создание *взаимодействующих с сервером* приложения с использованием Microsoft Silverlight. Silverlight предлагает ряд прикладных программных интерфейсов (APIs), обеспечивающих возможность работы с удаленными данными, и здесь мы тщательно рассмотрим их использование для подключения к размещенному на сервере сервису и извлечения данных.

Начнем с простейшего средства подключения в Silverlight, класса *WebClient*, и разберемся с переопределением простой функциональности с помощью классов *HttpWebRequest* (Веб-запрос HTTP) и *HttpWebRespose* (Веб-ответ HTTP). Кроме того, выполним тестовое подключение к существующим Веб-сервисам и сервисами Windows Communication Foundation (WCF), используя прокси-классы, сформированные Microsoft Visual Studio. Наконец, рассмотрим реализацию междоменного доступа в Silverlight-приложении с помощью файлов междоменной политики.

Для подключения к сервису, прежде всего, необходим сам сервис, поэтому обсуждение каждой из технологий включает создание сервиса, который будет обрабатывать запросы и возвращать данные.

Создание сервиса данных

Некоторые примеры в данной главе демонстрируют подключение к сервису, который обеспечивает передачу набора финансовых данных через разные схемы, включая разновидности XML-over-HTTP (передача XML по HTTP) такие как HTTP-GET(передача XML в HTTP-запросе GET) и HTTP-POST (передача XML в HTTP-запросе POST), а также Web- и WCF-сервисы, которые предоставляют эти данные.

Прежде всего, запустим Visual Studio и создадим новый проект ASP.NET Web Application. Назовем его PriceHistoryService (История изменения цены). Это продемонстрировано на рис. 12-1.

Таким образом, будет создан стандартное Веб-приложение ASP.NET, включающее Веб-форму Default.aspx. Прежде чем реализовывать прием некоторых параметров и возвращение набора данных, напишем вспомогательные методы для взаимодействия приложения с Веб- и WCF-сервисами.

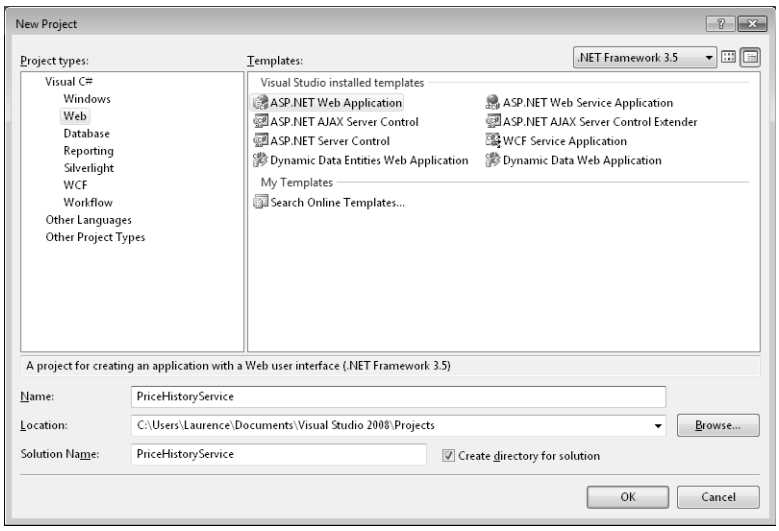

РИС. 12-1 Создание нового проекта PriceHistoryService.

Создаваемый сервис будет предлагать пользователю набор данных, предоставляемых Yahoo.

Сервис Yahoo возвращает CSV-файл (формат файла, в котором значения данных разделены запятыми), включающими следующие поля: *Date* (Дата), *Opening Price* (Начальная цена), *Closing Price* (Заключительная цена), *High* (Максимальная), *Low* (Минимальная), *Volume* (Объем) и *Adjusted Close* (Установленная цена закрытия). Вызываемый API предельно прост:

```
http://ichart.finance.yahoo.com/table.csv
```

Также могут использоваться параметры, представленные в табл. 12-1.

ТАБЛИЦА 12-1 **Параметры сервиса данных Yahoo**

Параметр	Значение
s	Символ акций (т.е. MSFT)
a	Начальный месяц (0 = Январь, 11 = Декабрь)
b	Начальный день
c	Начальный год
d	Конечный месяц (0 = Январь, 11 = Декабрь)
e	Конечный день
f	Конечный год
g	Всегда используйте букву d
ignore	Всегда используйте значение .csv

Получить хронологию изменения курса акций Microsoft (MSFT) с 1 января 2008 года по 1 января 2009 можно с помощью следующего URL:

http://ichart.finance.yahoo.com/table.csv?s=MSFT&a=0&b=1&c=2008&d=0&e=1&f=2009&g=d&ignore=.csv

Рассмотрим C#-метод, который принимает строковые параметры символа, начальной даты и конечной даты для построения этого URI:[1]:

```
using System.Text;
using System.Xml;
using System.Data;
using System.Net;
using System.IO;


public string BuildYahooURI(string strTicker,
  string strStartDate, string strEndDate)
  {
    string strReturn = "";
    DateTime dStart = Convert.ToDateTime(strStartDate);
    DateTime dEnd = Convert.ToDateTime(strEndDate);

    string sStartDay = dStart.Day.ToString();
    string sStartMonth = (dStart.Month -1).ToString();
    string sStartYear = dStart.Year.ToString();

    string sEndDay = dEnd.Day.ToString();
    string sEndMonth = (dEnd.Month - 1).ToString();
    string sEndYear = dEnd.Year.ToString();

    StringBuilder sYahooURI = new StringBuilder("http://ichart.finance.yahoo.com/table.csv?s=");
    sYahooURI.Append(strTicker);
    sYahooURI.Append("&a=");
    sYahooURI.Append(sStartMonth);
    sYahooURI.Append("&b=");
    sYahooURI.Append(sStartDay);
```

[1] Указанные ниже конструкции *using* нужно поместить в начало файла исходного кода (*прим. редактора*).

```
    sYahooURI.Append("&c=");
    sYahooURI.Append(sStartYear);
    sYahooURI.Append("&d=");
    sYahooURI.Append(sEndMonth);
    sYahooURI.Append("&e=");
    sYahooURI.Append(sEndDay);
    sYahooURI.Append("&f=");
    sYahooURI.Append(sEndYear);
    sYahooURI.Append("&g=d");
    sYahooURI.Append("&ignore=.csv");

    strReturn = sYahooURI.ToString();
    return strReturn;
}
```

Следующий шаг после создания URI для данных – чтение этих данных и их использование. В данном случае данные, разделенные запятыми, преобразовываются в XML. Рассмотрим метод, который делает это:

```
public XmlDocument getXML(string strTicker,
  string strStartDate, string strEndDate)
 {
   XmlDocument xReturn = new XmlDocument();
   DataSet result = new DataSet();
   string sYahooURI = BuildYahooURI(strTicker, strStartDate, strEndDate);
   WebClient wc = new WebClient();
   Stream yData = wc.OpenRead(sYahooURI);
   result = GenerateDataSet(yData);
   StringWriter stringWriter = new StringWriter();
   XmlTextWriter xmlTextwriter = new XmlTextWriter(stringWriter);
   result.WriteXml(xmlTextwriter, XmlWriteMode.IgnoreSchema);
   XmlNode xRoot = xReturn.CreateElement("root");
   xReturn.AppendChild(xRoot);
   xReturn.LoadXml(stringWriter.ToString());

   return xReturn;
}
```

Последний необходимый нам вспомогательный метод – *GenerateDataSet* (Сформировать набор данных), который вызывается в предыдущем фрагменте кода. Эта метод принимает строку разделенных запятыми значений, возвращенных сервисом данных Yahoo, и преобразует ее в *DataSet* (Набор данных). *DataSet* используется позже в Веб- и WCF-сервисах, но, на данный момент, он сериализован в XML.

```
public DataSet GenerateDataSet(Stream yData)
      {
         DataSet result = new DataSet();
         StreamReader sRead = new StreamReader(yData);
         string[] columns = sRead.ReadLine().Split(',');
         result.Tables.Add("TimeSeries");
         foreach (string col in columns)
         {
            // Удаляем пробелы из имен
            string thiscol = col.Replace(" ", "");
            // Добавляем имя столбца
            result.Tables["TimeSeries"].Columns.Add(thiscol);
         }
         string sData = sRead.ReadToEnd();
         string[] rows = sData.Split('\n');

         foreach (string row in rows)
         {
            string[] items = row.Split(',');
            result.Tables["TimeSeries"].Rows.Add(items);

         }
         return result;
      }
```

Поместите эти методы в класс *HelperFunctions* (Вспомогательные функции) и добавьте его в Веб-проект ASP.NET. Далее добавьте Веб-форму ASP.NET (ASPX) под именем GetPriceHistory и удалите из ASPX-страницы HTML-разметку, чтобы она выглядела следующим образом:

```
<%@ Page Language="C#" AutoEventWireup="true" CodeBehind="GetPriceHistory.aspx.cs"
Inherits="PriceHistoryService.GetPriceHistory" %>
```

Примечание Можете оставить объявление <!DOCTYPE> вверху страницы.

Замечательно то, что теперь можно создавать код для записи непосредственно в буфер ответов и задавать тип ответа, позволяющий передавать XML по HTTP.

Вероятно, вы заметили, что управление связью реализуется с помощью класса *WebClient*. Это *WebClient*, который является частью Microsoft .NET Framework, используемой ASP.NET. В следующем разделе будет рассмотрен *WebClient Silverlight*, который поразительно похож на этот!

Итак, вспомогательные методы принимают строковые значения символа акций, начальной и конечных дат. Эти строковые значения можно сделать параметрами ASPX, передать в рассмотренные выше вспомогательные методы для формирования XML, который затем записывается в буфер ответов. Кроме того, зададим тип MIME (Multipurpose Internet Mail Extensions – Многоцелевые расширения электронной почты Интернет) равным *text/xml*, чтобы любое средство чтения распознавало ответ как XML, а не как текст.

Рассмотрим код, который делает это. Помним, что *HelperFunctions* – это имя класса, содержащего методы для построения URI Yahoo, его чтения и преобразования формата с разделяющими запятыми в XML:

```
HelperFunctions hlp = new HelperFunctions();
protected void Page_Load(object sender, EventArgs e)
{
  string strTicker, strStartDate, strEndDate;

  if(Request.Params["ticker"]!=null)
    strTicker = Request.Params["ticker"].ToString();
  else
    strTicker = "MSFT";

  if(Request.Params["startdate"]!=null)
    strStartDate = Request.Params["startdate"].ToString();
  else
    strStartDate = "1-1-2008";

  if(Request.Params["enddate"]!=null)
    strEndDate = Request.Params["enddate"].ToString();
  else
    strEndDate = "1-1-2009";

  XmlDocument xReturn = hlp.getXML(strTicker, strStartDate, strEndDate);

  Response.ContentType = "text/xml";
  Response.Write(xReturn.OuterXml);

}
```

Убедитесь в наличии всех необходимых ссылок и соответствующих системных библиотек (например, вверху страницы кода обязательно должно присутствовать *using System.Xml*).

Теперь у нас имеется простой сервис для передачи XML по HTTP, возвращающий набор данных. Эти данные представлены на рис. 12-2.

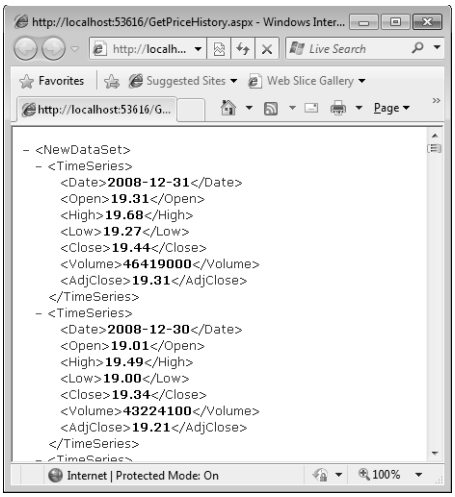

РИС. 12-2 Сервис для передачи XML по HTTP.

В следующем разделе будет показано, как реализуется доступ к этим данным в Silverlight-клиенте. Обратите внимание, что в этом проекте применяется сервер разработки ASP.NET, и в данном сеансе он использует порт 53616 (это можно увидеть на рис. 12-2). Это создает некоторые проблемы при использовании Silverlight для доступа к нему, потому что в Silverlight действуют междоменные ограничения при связи по сети. Итак, не забудьте на странице Веб-свойств проекта[1] изменить его настройки на использование локального сервера IIS (Internet Information Services)[2].

Использование класса *WebClient*

Чтобы реализовать базовую возможность взаимодействия по протоколу HTTP для получения данных через Веб, такую как передача XML по HTTP, класс *WebClient* подходит идеально. Он прост в использовании, прост в инициализации и обеспечивает хорошую производительность. Он также поддерживает основные операции с заголовками в случаях, если требуется доступ к HTTP-заголовкам, например, при использовании HTTP-форм.

В предыдущем разделе был создан сервис, обратиться к которому можно по следующему URL:

http://servername:serverport/GetPriceHistory.aspx?ticker=[ticker]&startdate=[startdate]&enddate=[enddate]

При работе с локального IIS можно опустить параметр *serverport* (порт сервера).

Создадим в Visual Studio новое приложение Silverlight. В меню File выберем New Project. В диалоговом окне New Project выбираем шаблон проекта Silverlight Application и применяем его для создания нового проекта под именем PHWebClient.

Примите предложение мастера проекта о размещении Silverlight-приложения на новом Веб-сайте и используйте при этом имя проекта по умолчанию.

Можно начать с очень простого примера, который будет выводить на экран исходные цены. Для описания области формирования визуального представления элементов используем элемент управления XAML *ItemsControl*. В нем определяем шаблон отображения этих данных. Для простоты расположим элементы управления *TextBlock* вертикально с помощью *StackPanel*.

Рассмотрим XAML-код:

---

[1] Свойства проекта можно открыть, кликнув правой кнопкой проект в Solution Explorer и выбрав пункт Properties контекстного меню (*прим. редактора*).

[2] Если при выполнении этой операции вы получите сообщение о нехватке прав, запустите Visual Studio с правами администратора (*прим. редактора*).

```
<ItemsControl x:Name="_items">
  <ItemsControl.ItemTemplate>
    <DataTemplate>
      <StackPanel Orientation="Vertical">
        <TextBlock FontWeight="Bold" Text="{Binding open}" />
      </StackPanel>
    </DataTemplate>
  </ItemsControl.ItemTemplate>
</ItemsControl>
```

Обратите внимание, что для свойства *Text* элемента управления *TextBlock* выполнена привязка (*Binding*) к значению *open*. Таким образом, пора заняться кодом Silverlight-клиента, который принимает данные и обеспечивает привязку данных.

Начнем с класса, представляющего набор данных. Как вы, вероятно, помните, набор данных включает следующие значения: дата торгов, начальная цена, конечная цена, максимальная цена, минимальная цена, объем и установленная цена закрытия. Вот класс, который может хранить эти значения:

```
public class TimeSeriesData
  {
    public DateTime date { get; set; }
    public double open { get; set; }
    public double close { get; set; }
    public double high { get; set; }
    public double low { get; set; }
    public double volume { get; set; }
    public double adjclose { get; set; }

    public TimeSeriesData(DateTime dteIn, double openIn, double closeIn,
          double highIn, double lowIn, double volumeIn, double adjCloseIn)
    {
      date = dteIn;
      open = openIn;
      close = closeIn;
      high = highIn;
      low = lowIn;
      volume = volumeIn;
      adjclose = adjCloseIn;
    }

    public TimeSeriesData()
    {
  }
}
```

Ничего особенного, простой класс для хранения данных. Но как подключиться к сервису данных истории изменения цены, получить данные и обеспечить связывание? Все предельно просто, если использовать *WebClient* в Silverlight.

WebClient определяет адрес с помощью экземпляра класса *System.Net.Uri*. Итак, рассмотрим код для создания одного из этих экземпляров под именем *uri*:

```
Uri uri = new Uri("http://localhost/PriceHistoryService/GetPriceHistory.aspx?ticker=MSFT&startdate=1-7-2009&enddate=1-10-
2009", UriKind.RelativeOrAbsolute);
```

Теперь создаем экземпляр класса *WebClient* и определяем метод, который будет вызываться при обратном вызове по завершении чтения данных сервиса:

```
WebClient wc = new WebClient();
wc.OpenReadCompleted += new OpenReadCompletedEventHandler(wc_OpenReadCompleted);
```

Осталось лишь прочитать возвращенное в результате вызова сервиса значение:

```
wc.OpenReadAsync(uri);
```

По завершении чтения данных сервиса вызывается метод-обработчик *wc_OpenReadCompleted*. Он принимает параметр *e* типа *OpenReadCompletedEventArgs*. Этот аргумент включает свойство *Result* типа

Stream. Чтение XML-данных из этого потока осуществляется с помощью *StreamReader* (Модуль чтения потока).

```
StreamReader read = new StreamReader(e.Result);
string strXml = read.ReadToEnd();
```

Из этой строки класс *XDocument* может создать XML-документ с помощью метода *Parse* (Выполнить синтаксический разбор):

```
XDocument xmlDoc = XDocument.Parse(strXml);
```

Теперь, когда мы имеем *XDocument* с данными, можем применить LINQ и создать перечисление *IEnumerable* класса *TimeSeriesData* (Хронологические данные), который был описан ранее. Silverlight использует интерфейс *IEnumerable* при связывании данных. Вот необходимый для этого запрос LINQ:

```
IEnumerable<TimeSeriesData> myTimeSeries = from item in xmlDoc.Descendants("TimeSeries")
        where item.Element("Open") != null
   select new TimeSeriesData
   {
    open = Convert.ToDouble(item.Element("Open").Value, new CultureInfo("en-US").NumberFormat)
   };
```

Обратите внимание, что данный код просто извлекает значение *Open* из XML. В последующих примерах демонстрируется извлечение всех остальных значений, но здесь это опущено для краткости.

Визуальный интерфейс нам обеспечивает *ItemsControl*. Связывание данных осуществляется, если его свойству *ItemsSource* задано значение объекта, реализующего *IEnumerable*. Однако если этот код выполняется в *потоке пользовательского интерфейса* (например, если клиент вызывается из конструктора *MainPage*, как в этом случае), необходимо использовать *Dispatcher* (Диспетчер), чтобы связывание происходило вне потока.

Это делается так:

```
Dispatcher.BeginInvoke(() => _items.ItemsSource = myTimeSeries);
```

Поскольку в этом приложении используется LINQ, не забудьте добавить ссылку на DLL System.Xml.Linq в папке References решения и соответствующее описание *using* вверху страницы кода. В противном случае, код не будет компилироваться.

Итак, сведем все вышеизложенное воедино и рассмотрим полный выделенный код для этого класса Silverlight:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Net;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Animation;
using System.Windows.Shapes;
using System.IO;
using System.Xml;
using System.Xml.Linq;
namespace PHWebClient
{
 public partial class MainPage : UserControl
 {
  public MainPage()
  {
   InitializeComponent();
   WebClient wc = new WebClient();
   Uri uri = new
```

```
Uri("http://localhost/PriceHistoryService/GetPriceHistory.aspx?ticker=MSFT&startdate=1-7-2009&enddate=1-10-2009",
UriKind.RelativeOrAbsolute);
    wc.OpenReadCompleted += new OpenReadCompletedEventHandler(wc_OpenReadCompleted);
    wc.OpenReadAsync(uri);
  }

  void wc_OpenReadCompleted(object sender, OpenReadCompletedEventArgs e)
  {
    StreamReader read = new StreamReader(e.Result);
    string strXml = read.ReadToEnd();
    XDocument xmlDoc = XDocument.Parse(strXml);
    IEnumerable<TimeSeriesData> myTimeSeries = from item in xmlDoc.Descendants("TimeSeries")

      where item.Element("Open") != null
      select new TimeSeriesData
      {
        open = Convert.ToDouble(item.Element("Open").Value

               , new CultureInfo("en-US").NumberFormat)
      };

    Dispatcher.BeginInvoke(() => _items.ItemsSource = myTimeSeries);
  }
 }
}
```

На рис. 12-3 представлен результат выполнения этого кода.

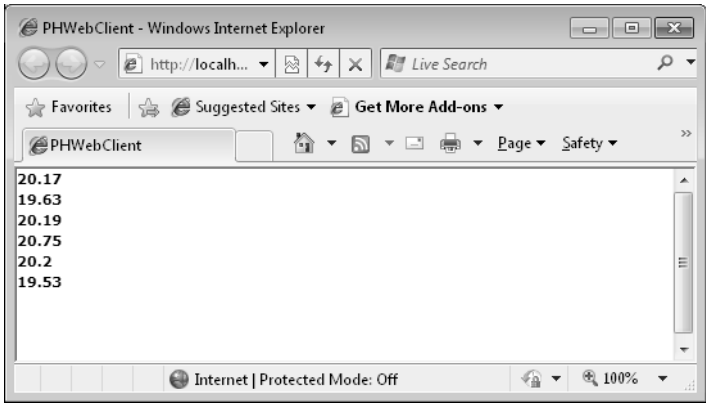

РИС. 12-3 Формирование визуального представления данных с помощью *WebClient*.

В данном разделе вы увидели, как просто подключиться к сервису данных, который обеспечивает передачу XML по HTTP, и выполнить простое связывание данных. Эта схема лежит в основе многих сценариев, но давайте копнем немного глубже. Некоторые сервисы данных требуют передачи значений методом POST в HTTP-заголовке с использованием HTTP-форм. В следующем разделе сначала создадим простой сервер, принимающий параметры HTTP-запроса POST, а затем построим клиент, их использующий.

Расширение сервиса для обработки HTTP-запросов POST

Ранее вы создали решение PriceHistoryService, включающее страницу GetPriceHistory.aspx, которая использовалась для получения истории изменения цены с сервиса, обеспечивающего передачу XML по HTTP. Это решение можно расширить и создать сервис, который будет обрабатывать HTTP-формы с переменными HTTP-POST. Вернитесь к имеющемуся решению, добавьте в него универсальный обработчик (ASHX) и назовите его GetPriceHistoryWithPOST.ashx.

В выделенном коде можно увидеть обработчик события *ProcessRequest* (Обработать запрос), принимающий *HttpContext* (HTTP-контекст) как параметр. Это все, что необходимо для перехвата параметров, передаваемых с HTTP-запросом POST, использования их для вызова сервиса Yahoo, получения данных и форматирования их в виде XML для клиента, чтобы он мог обработать их.

Рассмотрим код, обеспечивающий это. Он очень похож на ASPX-код, рассматриваемый в данной главе ранее:

```
public void ProcessRequest(HttpContext context)
{
 HelperFunctions hlp = new HelperFunctions();
 string strTicker;
 string strStartDate;
 string strEndDate;
 if (context.Request.HttpMethod == "POST")
 {
  if (context.Request.Form["ticker"] != null)
   strTicker = context.Request.Form["ticker"].ToString();
  else
   strTicker = "MSFT";

  if (context.Request.Form["startdate"] != null)
   strStartDate = context.Request.Form["startdate"].ToString();
  else
   strStartDate = "1-1-2008";

  if (context.Request.Form["enddate"] != null)
   strEndDate = context.Request.Form["enddate"].ToString();
  else
   strEndDate = "1-1-2009";

  XmlDocument xReturn = hlp.getXML(strTicker, strStartDate, strEndDate);

  context.Response.ContentType = "text/xml";
  context.Response.Write(xReturn.OuterXml);

 }
 else
  {
   context.Response.Write("This must be called with an HTTP-POST with params in the header");
  }

 }
```

Не забудьте добавить выражения *using* для System.Xml и System.Web.Services в начало кода, чтобы обеспечить его компиляцию.

Этот код просто проверяет, был ли выполнен HTTP-запрос POST. Если запрос поступил, из HTTP-формы извлекаются значения параметров символа акций, начальной даты и конечной даты. Затем все эти данные передаются в метод *getXML* класса *HelperFunctions*, где выполняется вся тяжелая работа.

После этого данные записываются в поток ответов. Если вызвать этот ASHX без запроса POST HTTP-формы, в поток ответов записывается текст ошибки.

Использование HttpWebRequest и HttpWebResponse

Классы *HttpWebRequest* и *HttpWebResponse* обеспечивают тонкое управление обменом данными по HTTP, поэтому могут использоваться для реализации HTTP-запроса POST к серверу.

Их применение может показаться немного запутанным на первый взгляд, но вскоре вы освоитесь. Схема такова:

1. Создаем и инициализируем новый *HttpWebRequest*.
2. Получаем поток *Request* (Запрос) асинхронного обратного вызова.
3. Когда запрос готов, пишем параметры в него и настраиваем обратный вызов для ответа на этот запрос.
4. При поступлении ответного обратного вызова, извлекаем из него необходимые данные.

Итак, для начала создаем новый проект Silverlight PHPostClient. Добавим в файл MainPage.xaml класс *ItemsControl* (Компонент отображения набора элементов), как делали ранее в проекте *WebClient*. Это должно выглядеть так:

```
<UserControl x:Class="PHPostClient.MainPage"
  xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
  Width="640" Height="480">
  <Grid x:Name="LayoutRoot" Background="White">
    <ItemsControl x:Name="_items">
      <ItemsControl.ItemTemplate>
        <DataTemplate>
          <StackPanel Orientation="Vertical">
            <TextBlock FontWeight="Bold" Text="{Binding open}" />
          </StackPanel>
        </DataTemplate>
      </ItemsControl.ItemTemplate>
    </ItemsControl>
  </Grid>
</UserControl>
```

После этого настраиваем связь в файле выделенного кода. Сначала инициализируем все в конструкторе *MainPage()*. Как и ранее, не забываем добавлять ссылку проекта на DLL System.Xml.Linq и выражения для System.IO и System.Xml.Linq в начало страницы кода.

Чтобы настроить *HttpWebRequest*, необходим URI вызываемого сервиса:

```
Uri uri = new Uri("http://localhost/PriceHistoryService/GetPriceHistorywithPOST.ashx");
HttpWebRequest request = (HttpWebRequest)HttpWebRequest.Create(uri);
```

В объекте запроса требуется указать, что это запрос POST, а также сообщить в свойстве *ContentType* (Тип содержимого), что это HTTP-форма:

```
request.Method = "POST";
request.ContentType = "application/x-www-form-urlencoded";
```

Наконец, начинаем инициализацию потока запроса. Он выполняется асинхронно, поэтому необходимо определить метод обратного вызова для обеспечения обработки следующего шага после установления потока запроса. В данном случае используется метод обратного вызова *RequestProceed* (Обработка запроса):

```
request.BeginGetRequestStream(new AsyncCallback(RequestProceed), request);
```

Метод обратного вызова *RequestProceed* должен принимать параметр типа *IASyncResult*. Рассмотрим пример, в котором этот параметр называется *asyncResult* (Результат асинхронной операции):

```
void RequestProceed(IAsyncResult asyncResult)
 {
 }
```

Переменная *HttpRequest* не глобальная, поэтому в этом методе нет контекста для ее получения. Но она хранится в свойстве *AsyncState* (Состояние асинхронной операции) переменной *asyncResult*, т.е. выйти из этой ситуации можно так:

```
HttpWebRequest request = (HttpWebRequest)asyncResult.AsyncState;
```

Следующий шаг после этого – записать данные как параметры запроса, чтобы они могли быть переданы в сервис. Это делается с помощью *StreamWriter* (Модуль записи в поток). Поток доступен из самого объекта запроса:

```
StreamWriter postDataWriter = new StreamWriter(request.EndGetRequestStream(asyncResult));
```

Запись параметров состоит просто в записи строк в этот *StreamWriter*:

```
postDataWriter.Write("ticker=MSFT");
postDataWriter.Write("&startdate=1-7-2009");
```

```
postDataWriter.Write("&enddate=1-10-2009");
postDataWriter.Close();
```

Итак, все необходимые данные переданы в запрос, теперь осталось лишь начать слушать ответ. Для этого опять используем обратный вызов. Вызывается метод запроса *BeginGetReponse* (Начать слушать ответ) и в него передается имя метода обратного вызова:

```
request.BeginGetResponse(new AsyncCallback(ResponseProceed), request);
```

Ну вот, практически, все готово! Заключительный шаг – принять ответ и прочитать данные из него. Используемый метод обратного вызова называется *ResponseProceed* (Обработка ответа), и он также принимает параметр *IASyncResult*:

```
void ResponseProceed(IAsyncResult asyncResult)
{}
```

Как и ранее, из *asyncResult* необходимо извлечь ссылку на запрос, и, кроме того, обратиться к ответу можно из объекта запроса:

```
HttpWebRequest request = (HttpWebRequest)asyncResult.AsyncState;
HttpWebResponse response = (HttpWebResponse)request.EndGetResponse(asyncResult);
```

Получив объект ответа, можно читать данные из него с помощью *StreamReader*:

```
StreamReader responseReader = new StreamReader(response.GetResponseStream());
string responseString = responseReader.ReadToEnd();
```

Ну вот, мы и вернулись на знакомую территорию. У нас имеется строка XML-данных, которую можно загрузить в *XDocument*, с помощью LINQ извлечь необходимые данные в *IEnumerable* и связать его со своим пользовательским интерфейсом. Не забудьте добавить в свой проект класс *TimeSeriesData*.

```
XDocument xReturn = XDocument.Parse(responseString);
IEnumerable<TimeSeriesData> myTimeSeries = from item in xReturn.Descendants("TimeSeries")

    where item.Element("Open") != null
  select new TimeSeriesData
  {
   open = Convert.ToDouble(item.Element("Open").Value,

         new CultureInfo("en-US").NumberFormat)
  };

  Dispatcher.BeginInvoke(() => _items.ItemsSource = myTimeSeries);
```

Результаты выполнения этого кода представлены на рис. 12-4.

РИС. 12-4 Визуальное представление данных, возвращенных HTTP-запросом POST.

Очень похоже на вывод, представленный на рис. 12-3, потому что уровень визуализации никак не изменился, а вот для получения данных используется другой протокол связи! В следующем разделе рассмотрим другую технологию обмена информацией с сервисами на сервере – Веб-сервисы SOAP.

Веб-сервис SOAP

В предыдущих разделах были созданы простые сервисы передачи XML по HTTP и HTTP-POST, возвращающие данные Silverlight-клиенту. Существует еще один способ реализации сервиса данных – Веб-сервис SOAP. Его построению и использованию посвящен данный раздел.

Первым делом, вернемся к решению PriceHistoryService и добавим в него новый Веб-сервис *PriceHistoryWebService*. Используем для него расширение .asmx.

Веб-сервис – это особый тип сервиса HTTP-POST, который для передачи структурированных сообщений использует Простой протокол доступа к объектам (Simple Object Access Protocol, SOAP). Visual Studio обрабатывает все базовые методы за вас, поэтому создание Веб-сервиса не представляет никакого труда.

В файле PriceHistoryWebService.asmx.cs можно увидеть базовый метод *HelloWorld*, который обозначен как *WebMethod* (Веб-метод) следующим образом:

```
 [WebMethod]
public string HelloWorld()
{
  return "Hello World";

}
```

Если использовать вспомогательные методы, описанные в этой главе ранее, создать Веб-метод для возвращения набора данных не представляет труда. Просто создаем метод, который принимает три параметра (символ акций, начальную дату и конечную дату) и применяет вспомогательный метод *getXML* для получения *XMLDocument*, который затем возвращается вызывающей стороне.

Рассмотрим код:

```
 [WebMethod]
public XmlDocument getPriceHistoryAsXML(string strTicker, string strStartDate, string strEndDate)
  {
    XmlDocument xReturn = new XmlDocument();    HelperFunctions hlp = new HelperFunctions();


    xReturn = hlp.getXML(strTicker, strStartDate, strEndDate);
    return xReturn;

  }
```

Выполнение этого кода приводит в окружение тестирования SOAP-сервисов, которое представлено на рис. 12-5.

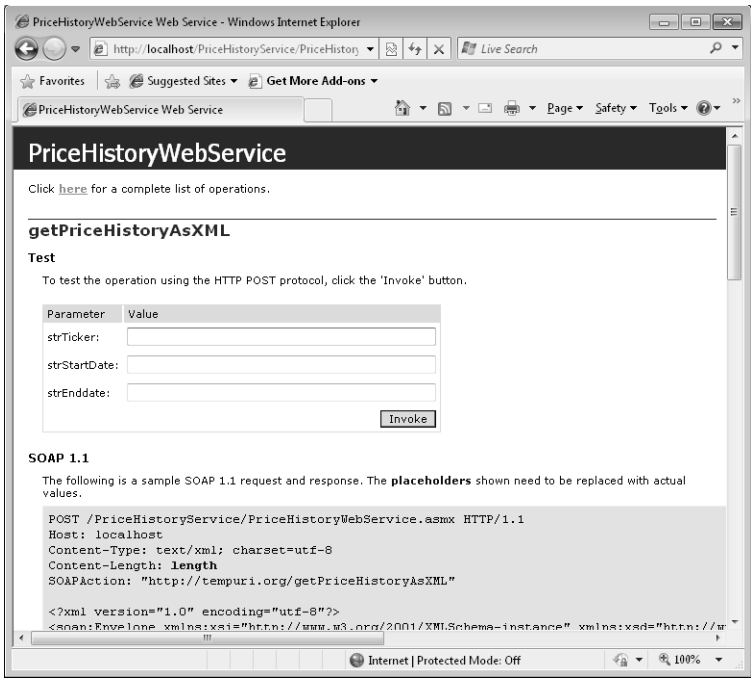

РИС. 12-5 Окружение тестирования Веб-сервисов.

Это окружение позволяет вводить значения параметров и просматривать получаемые результаты по нажатию кнопки Invoke (Запустить), а также предоставляет ссылку вверху страницы для получения адреса файла WSDL (Web Services Description Language – язык описания Веб-сервисов), который описывает данный Веб-сервис. Адрес этого WSDL выглядит примерно так:

http://localhost/PriceHistoryService/PriceHistoryWebService.asmx?WSDL

Обратите внимание на этот адрес, потому что он понадобиться позже при создании используемого Silverlight прокси-класса Веб-сервиса.

Создание клиента Веб-сервисов в Silverlight

Теперь, имея Веб-сервис и адрес его описания в формате WSDL, можно приступать к созданию клиента, использующего этот Веб-сервис.

Создадим новое приложение Silverlight и назовем его PHSoapClient.

Перейдите к проекту Silverlight и выберите References. Щелкните правой кнопкой мыши и выберите опцию Add Service Reference. Это обеспечит открытие диалогового окна Add Service Reference, как показано на рис. 12-6.

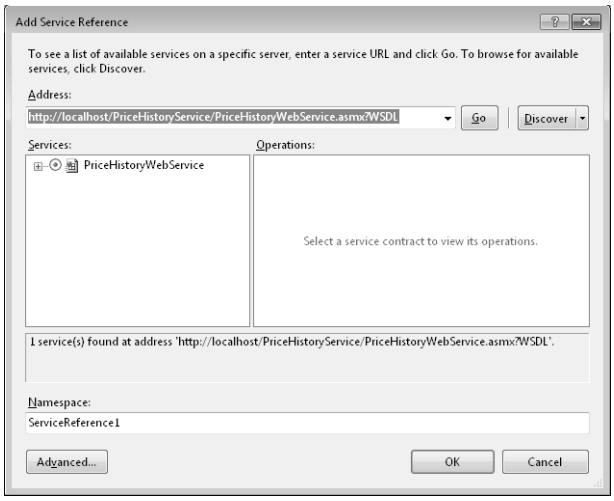

РИС. 12-6 Добавление Веб-ссылки.

В поле Address (Адрес) введите адрес WSDL-файла и нажмите Go. Заданный сервис будет найден и пространство имен по умолчанию создано (ServiceReference1). Измените имя на что-то более понятное, скажем, PHWebService, и щелкните OK. Это обеспечит создание прокси-класса Веб-сервиса.

Этот прокси-класс включает SOAP-клиент, который может использоваться для связи с сервисом. Клиент называется *<Имя Сервиса>*SoapClient. В данном случае экземпляр этого класса может быть объявлен с помощью следующего кода:

```
PHWebService.PriceHistoryWebServiceSoapClient myClient =
   new PHWebService.PriceHistoryWebServiceSoapClient();
```

Чтобы вызвать сервис и возвратить данные, просто описываем метод обратного вызова и вызываем его. Прокси-класс добавляет *Async* в конец имени метода, поэтому для вызова сервиса с Веб-методом *foo* используется метод *fooAsync*.

Рассмотрим код для настройки обратного вызова и вызова Веб-сервиса для получения некоторого набора данных:

```
myClient.getPriceHistoryAsXMLCompleted +=
 new EventHandler<PHWebClient.PHWebService.getPriceHistoryAsXMLCompletedEventArgs>
 (myClient_getPriceHistoryAsXMLCompleted);
myClient.getPriceHistoryAsXMLAsync("MSFT", "1-7-2009", "1-11-2009");
```

Обратите внимание, что клавиша Tab клавиатуры позволяет автоматически завершать код этих методов.

Ваш метод обратного вызова будет получать обратный вызов с параметром <имя Веб-метода + *Аргументы завершенного события*>. Итак, в данном случае, при вызове *getPriceHistoryAsXML* возвращается объект *e* типа *getpriceHistoryAsXMLCompletedEventArgs*.

Возвращенный XML сериализован в строку, поэтому может быть загружен в *XDocument*:

```
XDocument xmlDoc = XDocument.Parse(e.Result.ToString());
```

Теперь можно создать *IEnumerable* с помощью LINQ-запроса, точно так же как это делалось в других примерах данной главы. Выполните привязку *IEnumerable* к *ItemsPresenter*, как раньше:

```
IEnumerable<TimeSeriesData> myTimeSeries = from item in xmlDoc.Descendants("TimeSeries")

    where item.Element("Open")!= null
 select new TimeSeriesData
 {
  open = Convert.ToDouble(item.Element("Open").Value,
```

```
        new CultureInfo("en-US").NumberFormat)
  };
  Dispatcher.BeginInvoke(() => _items.ItemsSource = myTimeSeries);
```

Как видите, использование Веб-сервиса способствует упрощению задачи по обмену данными за счет автоматически создаваемого прокси-класса, и в Silverlight есть встроенная поддержка для этого.

Однако заметьте, что при использовании в Веб-сервисе классов, не поддерживаемых Silverlight, создание, компиляция или выполнение прокси может дать сбой. Например, проблемы в прокси Silverlight возникают, если Веб-сервис возвращает *Dataset*.

В следующем разделе обсуждается WCF. WCF обеспечивает аналогичные возможности связи, но при этом структурирование данных и более высокую производительность.

WCF-сервис

До этого мы только вносили изменения в проект сервиса. В данном случае создадим новый сервис. Зачастую смешение WCF и не-WCF в одном решении требует применения некоторых хитростей, особенно в файлах конфигурации, поэтому не будем усложнять и создадим новое приложение WCF-сервиса. В Visual Studio выберите меню File, щелкните New Web Site и из предлагаемого списка доступных шаблонов выберите WCF Service. Назовите решение PHWCFService.

Прежде всего, добавим экземпляр класса *TimeSeriesData* и экземпляр класса *HelperFunctions*, с которыми мы уже работали в этой главе.

Поскольку класс *TimeSeriesData* обеспечивает взаимодействие между сервисом и клиентом, необходимо внести в него небольшие изменения и определить контракт данных между сервером и клиентом. Благодаря контракту данных прокси-генератор формирует на клиентской стороне классы, дублирующие классы, доступные на стороне сервера.

```
public class TimeSeriesData
  {
  [DataMember]
    public DateTime date { get; set; }
  [DataMember]
    public double open { get; set; }
  [DataMember]
    public double close { get; set; }
  [DataMember]
    public double high { get; set; }
  [DataMember]
    public double low { get; set; }
  [DataMember]
    public double volume { get; set; }
  [DataMember]
    public double adjclose { get; set; }

  public TimeSeriesData(DateTime dteIn, double openIn, double closeIn,
       double highIn, double lowIn, double volumeIn, double adjCloseIn)
  {
    date = dteIn;
    open = openIn;
    close = closeIn;
    high = highIn;
    low = lowIn;
    volume = volumeIn;
    adjclose = adjCloseIn;
  }

  public TimeSeriesData()
  {
  }
}
```

Как видите, единственное отличие между этим и используемым ранее описанием класса *TimeSeriesData* в том, что в данном случае открытые переменные-члены определяются как элементы данных с помощью атрибута *[DataMember]*.

Теперь добавим в проект новый сервис. В диалоговом окне Add New Item присваиваем сервису имя PHService. В решение добавляются три новых файла: PHService.svc, PHService.svc.cs и IPHService.cs.

Начнем с IPHService (I означает *interface*) и редактируем его следующим образом:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Runtime.Serialization;
using System.ServiceModel;
using System.Text;

namespace PHWCFService
{
    // ПРИМЕЧАНИЕ: Если изменяете имя интерфейса "IPHService" здесь, необходимо также обновить ссылку на
"IPHService" в Web.config.
    [ServiceContract]
    public interface IPHService
    {
        [OperationContract]
        IEnumerable<TimeSeriesData> getTimeSeries(string ticker, string startdate, string enddate);

    }
}
```

Здесь определяется интерфейс, используемый сервисом для обмена данными. Далее необходимо реализовать его. Это выполняется в файле PHService.svc.cs.

Рассмотрим код:

```
namespace PHWCFService
{
  public class PHService : IPHService
  {
    HelperFunctions hlp = new HelperFunctions();
    public IEnumerable<TimeSeriesData> getTimeSeries(string ticker, string startdate, string enddate)
    {
      XmlDocument xmlDoc = hlp.getXML(ticker, startdate, enddate);
      XDocument xDoc = XDocument.Parse(xmlDoc.InnerXml);
      IEnumerable<TimeSeriesData> ret = from item in xDoc.Descendants("TimeSeries")

        where item.Element("Open")!=null
        select new TimeSeriesData
        {
          adjclose = Convert.ToDouble(item.Element("AdjClose").Value, new CultureInfo("en-US").NumberFormat),
          close = Convert.ToDouble(item.Element("Close").Value, new CultureInfo("en-US").NumberFormat),
          high = Convert.ToDouble(item.Element("High").Value, new CultureInfo("en-US").NumberFormat),
          low = Convert.ToDouble(item.Element("Low").Value, new CultureInfo("en-US").NumberFormat),
          open = Convert.ToDouble(item.Element("Open").Value, new CultureInfo("en-US").NumberFormat),
          volume = Convert.ToDouble(item.Element("Volume").Value, new CultureInfo("en-US").NumberFormat),
          date = Convert.ToDateTime(item.Element("Date").Value, new CultureInfo("en-US").NumberFormat)
        };
      return ret;
    }
  }
}
```

До сих пор все кажется знакомым. В этом очарование последовательности .NET: навыки, приобретенные на одном уровне, распространяются на все остальные. Здесь мы видим, что бизнес-логика использования XML и LINQ на стороне сервера и на стороне клиента идентична!

Прежде чем приступать к созданию Silverlight-клиента, необходимо внести изменения в конфигурацию сервиса, чтобы Silverlight мог использовать его. WCF поддерживает множество типов связывания для

обмена данными, в отличие от Silverlight, поэтому требуется изменить соответствующие настройки конечной точки. Найдем в файле Web.config раздел <*system.serviceModel*>, в нем должны располагаться ваши сервисы. Тег <*service*> имеет атрибут связывания. Задайте ему значение *basicHttpBinding*, в противном случае, прокси Silverlight даст сбой.

После настройки сервиса можно приступать к созданию клиента. Добавьте в это решение проект Silverlight и назовите его PHWCFClient.

Теперь в проект Silverlight можно добавить ссылку на WCF-сервис. Для этого щелкните правой кнопкой мыши папку References и выберите Add Service Reference. Сервис можно найти с помощью кнопки Discover (Найти). Выберите сервис и присвойте ему удобное имя, например, PHService.

После этого Visual Studio создает в проекте Silverlight прокси-класс, который может обмениваться данными с WCF-сервисом.

Создание экземпляра клиента не представляет никакой сложности: имя сервиса является пространством имен прокси-класса, и именование прокси-класса выполняется по схеме *ИмяСервиса* + *Client*. В данном случае код выглядит следующим образом:

```
PHService.PHServiceClient myService = new PHService.PHServiceClient();
```

Теперь, как и во всех других примерах данной главы, создаем метод обратного вызова, который вызывается по завершении передачи данных и обращаемся к сервису, передавая необходимые параметры. Рассмотрим код:

```
myService.getTimeSeriesCompleted +=
  new EventHandler<PHWCFClient.PHService.getTimeSeriesCompletedEventArgs>
  (myService_getTimeSeriesCompleted);
myService.getTimeSeriesAsync("MSFT", "1-7-2009", "1-11-2009");
```

Не забывайте, что Visual Studio действительно облегчает задачу по вводу кода, пользуйтесь клавишей Tab.

В предыдущих примерах выполнялась передача XML, поэтому приходилось загружать XML в *XDocument*, чтобы получить возможность использовать LINQ для формирования *IEnumerable* для связывания. В данном случае, если посмотреть внимательней, можно заметить, что сервис возвращает непосредственно *IEnumerable*. То есть теперь для связывания возвращаемых данных необходимо лишь связать свойство *Result* аргументов события. Вот полный метод обратного вызова:

```
void myService_getTimeSeriesCompleted(object sender,
    PHWCFClient.PHService.getTimeSeriesCompletedEventArgs e)
{
  Dispatcher.BeginInvoke(() => _items.ItemsSource = e.Result);
}
```

Как и ранее, выполняется связывание данных, и отображается цена на момент закрытия торгов.

Междоменные вызовы

Во всех примерах данной главы вызовы выполнялись в рамках одного домена, вероятно, *http://localhost*. Однако нередко требуется вызывать данные с сервисов, размещающихся на других компьютерах. Для реализации этого Silverlight использует политики, т.е. вызываемый сервис должен разрешить доступ к нему. Выполняется это посредством размещения файла политики в корневом домене. Когда Silverlight видит обращение к данному домену, он первым делом ищет соответствующий файл политики в корне Веб-узла. Если файла политики там нет, вызов не выполняется и формируется ошибка. Если файл политики есть, Silverlight выполняет его синтаксический разбор. Если вызов запрещен политикой, вызов не выполняется и формируется

ошибка. В противном случае, Silverlight делает попытку вызова. Файл политики должен называться clientaccesspolicy.xml.

Рассмотрим пример файла политики:

```
<?xml version="1.0" encoding="utf-8" ?>
<access-policy>
  <cross-domain-access>
    <policy>
      <allow-from http-request-headers="*">
        <domain uri="*" />
      </allow-from>
      <grant-to>
        <resource path="/" include-subpaths="true" />
      </grant-to>
    </policy>
  </cross-domain-access>
</access-policy>
```

Узлы *allow-from* (разрешить от) определяют домены, вызовы от которых разрешены. В этом случае вместо конкретного URI задан символ *, свидетельствующий о том, что данный сервис общедоступен. Кроме того, в разделе *grant-to* (предоставить), в котором определяются путь и вложенные пути ресурса, можно задать доступные ресурсы сервера. В представленном здесь файле политики буквально говорится, что все, кто угодно, имеют доступ ко всему, что угодно.

Заключение

В данной главе рассматриваются различные методы подключения Silverlight к Веб-сервисам данных. Было представлено четыре основных метода взаимодействия с данными: *WebClient*, который обеспечивает базовую функциональность; *HttpWebRequest/HttpWebRespose*, которые могут использоваться для более тонкого управления, например, передачи и ответа на протоколы HTTP-форм; стандартный подход с применением Веб-сервисов SOAP, при котором с помощью файла Описания Веб-сервиса формируется прокси-класс, обеспечивающий возможность обмена данными с Веб-сервисом; и, наконец, WCF, который решает задачу по передаче данных просто и элегантно.

Глава 13

Стили и шаблоны в Silverlight

Microsoft Silverlight предлагает множество элементов управления, они были рассмотрены в нескольких предыдущих главах, а также возможность создания собственных элементов управления. Все поставляемые элементы управления Silverlight уже по умолчанию выглядят гармонично и насыщенно, но если возникнет необходимость изменить их стандартный вид соответственно собственному дизайну, Silverlight предлагает мощные возможности стилевого оформления и шаблонов, которые могут использоваться для настройки представления вашего приложения.

Понимание стилевого оформления

Чтобы понять, что такое стилевое оформление, рассмотрим простой пример использования кнопки в Silverlight-приложении. Далее представлен XAML для элемента управления *Button*:

```
<Canvas Background="LightBlue">
  <Button x:Name="btn" Content="Click Me!" Width="140"></Button>
</Canvas>
```

Это обеспечит отображение кнопки в стандартном стиле Silverlight (серебристого цвета, как будто с подсветкой сверху), как можно увидеть на рис. 13-1.

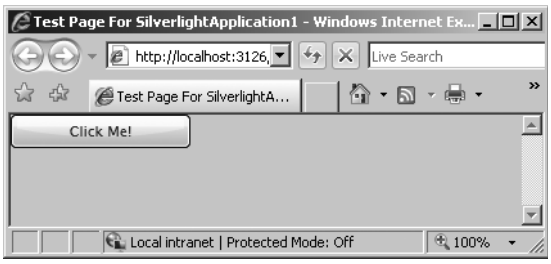

РИС. 13-1 Базовая кнопка Silverlight.

Как было показано в Главе 8, «Основные элементы управления Silverlight», и в Главе 9, «Дополнительные элементы управления Silverlight», многие элементы управления, включая *Button*, являются *элементами управления с содержимым*, т.е. могут формировать визуальное представление XAML-содержимого. Например, элемент управления *Button* может включать кое-что более интересное, чем просто строка «Click Me!» (Щелкни меня!). В элементы управления с содержимым можно добавлять некоторую функциональность, обеспечивающую пользователю более широкие возможности при взаимодействии с этим элементом управления.

Например, создадим кнопку с изображением. Для этого используем изображение как содержимое кнопки. Например:

```
<Canvas Background="LightBlue">
<Button Canvas.Left="40" Canvas.Top="40"
 x:Name="btn" Width="60" Height="80">
    <Button.Content>
      <StackPanel Orientation="Vertical">
        <Image Source="icon.jpg"
           Height="48" Width="48">
        </Image>
        <TextBlock Text="Click!"></TextBlock>
      </StackPanel>
    </Button.Content>
  </Button>
</Canvas>
```

Как это выглядит на экране, представлено на рис. 13-2.

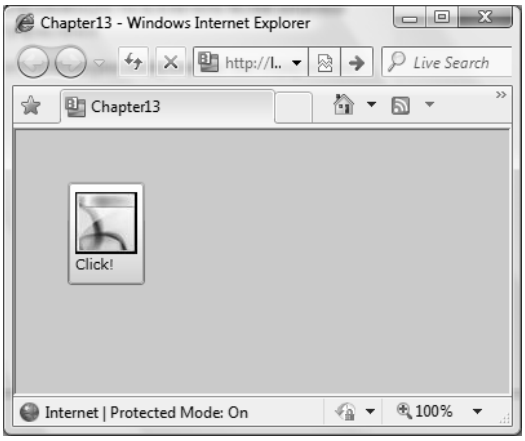

РИС. 13-2 Кнопка, использующая XAML-содержимое.

Вас не устраивает стандартный шрифт и цвет надписи на кнопке, и вы хотите еще больше изменить ее внешний вид? Сделать это в XAML довольно просто. Можно задать свойства, включающие семейство шрифтов, размер, насыщенность и т.д., настраивающие внешний вид кнопки. Например, посмотрим на обновленный *TextBlock*:

```
<TextBlock Text="Click!" FontFamily="Comic Sans Ms"
Foreground="MediumBlue" FontSize="20">
</TextBlock>
```

Также можно добавить несколько других реализаций кнопки. В XAML это сделать довольно просто, потому что можно просто скопировать первую кнопку, а затем задать свойства расположения для размещения копий на странице. Пример создания множества экземпляров кнопки представлен на рис. 13-3.

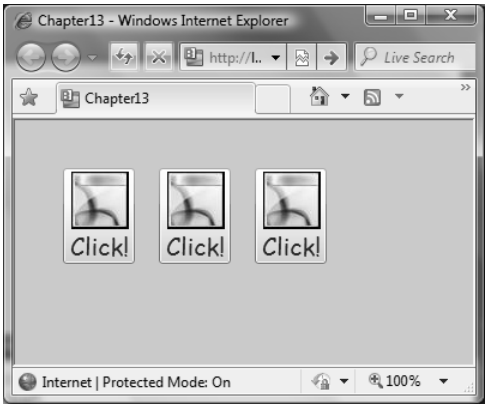

РИС. 13-3 Несколько экземпляров элемента управления *Button*.

Выглядит славно и очень просто реализуется, *но* использование такого подхода приведет к разрастанию XAML и, что еще хуже, большому количеству повторяющегося кода, что усложнит проверку, отладку, обслуживание и обновление приложения.

Возьмем элементы управления *TextBlock*. Для каждого из них явно объявлены свойства *FontFamily*, *Foreground* и *FontSize*, несмотря на то, что всем им задано одно и то же значение. А что, если требуется изменить какое-то из этих значений. Придется просматривать каждый элемент управления один за другим, чтобы обновить их все.

Вот где возникает необходимость в стилях. В следующих разделах рассмотрим, как вынести некоторые из этих данных в стиль.

Создание стиля

Стиль создается в разделе <*Resources*> (Ресурсы) контейнера. Здесь может быть создан один или более стилей, для которых задается имя стиля и тип объекта, к которому будет применяться стиль. Например, чтобы создать стиль *TextBlockStyle* (Стиль текстового блока) для элемента управления *TextBlock*, используется такой XAML:

```
<Style TargetType="TextBlock" x:Key="TextBlockStyle">
</Style>
```

Для стиля задается тип элемента управления, к которому он может применяться. Однако то, что этот стиль предполагается для *TextBlock*, не означает, что все экземпляры *TextBlock* будут использовать его. Вы не ограничены одним стилем для одного типа. В стиле должен быть задан тип элемента управления, для которого предназначен стиль, а также может быть задан *Key* (Ключ) стиля. Таким образом, можно иметь множество разных описаний стилей для элемента управления *TextBlock* с разными *Key*. Тогда, задавая свойство *Style* (Стиль) для разных *TextBlock*, вы просто выбираете тот стиль, который желаете применить к тому или иному *TextBlock*, по имени. Это будет продемонстрировано через мгновение, но сначала рассмотрим настройку стиля свойств *TextBlock*.

Для этого используется *Setter* (Установщик). *Setter* – это тег XAML, определяющий *свойство* и *значение*, которое необходимо задать этому свойству.

Далее представлен XAML, используемый ранее для задания свойств *Text, FontFamily, Foreground* и *FontSize* элемента *TextBlock*:

```
<TextBlock Text="Click!" FontFamily="Comic Sans Ms"
Foreground="MediumBlue" FontSize="20">
</TextBlock>
```

И вот теперь стиль, который делает то же самое:

```
<Style TargetType="TextBlock" x:Key="TextBlockStyle">
  <Setter Property="FontFamily" Value="Comic Sans Ms"></Setter>
  <Setter Property="Text" Value="Click!"></Setter>
  <Setter Property="Foreground" Value="MediumBlue"></Setter>
  <Setter Property="FontSize" Value="20"></Setter>
</Style>
```

Этот стиль должен быть определен на странице как ресурс. Как видите, корневым элементом XAML на странице является *UserControl*. У него есть дочерний элемент <*UserControl.Resources*>. Поместите тег <*Style*> в него.

Применяется этот стиль к *TextBlock* посредством свойства *Style*. Поскольку стиль определен как ресурс, для задания имени стиля используется синтаксис с ключевым словом *StaticResource* (Статический ресурс):

```
<TextBlock Style="{StaticResource TextBlockStyle}">
</TextBlock>
```

Теперь все свойства, применяемые к элементам управления *TextBlock* кнопки, задаются в одном месте, что значительно упрощает обслуживание и делает XAML удобным для восприятия и понимания.

Любое из свойств, определенных стилем, можно переопределить, просто задавая его в самом объекте. Например, требуется переопределить свойство *Foreground*, заданное стилем *MediumBlue* (Светло-синий). Сделать это очень просто, как показано в примере:

```
<TextBlock Style="{StaticResource TextBlockStyle}"
        Foreground="Black">
</TextBlock>
```

В этом случае, заданное здесь свойство *Foreground* переопределяет настройку стиля, и теперь для текста *TextBlock* будет использоваться черный цвет.

Также обратите внимание, что хотя стиль в данном примере определен для *TextBlock*, располагающегося в кнопке с изображением, он может применяться к любому *TextBlock* в данном *UserControl*.

Изменение области действия стиля

В предыдущем примере стиль создавался в *UserControl*, содержащем элементы управления. Это ограничивает область действия этого стиля данным *UserControl*. Если в приложении имеется несколько страниц, пришлось бы определять стили на каждой страницы, что весьма неэффективно.

К счастью, Silverlight позволяет описывать стили для всего приложения, задавая их в *App.xaml*. В стандартном файле *App.xaml*, создаваемом шаблоном Microsoft Visual Studio, уже определен раздел *<Resources>*!

```
<Application  xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
 xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
 x:Class="SilverlightApplication1.App">
 <Application.Resources>

 </Application.Resources>

</Application>
```

Итак, размещенные здесь описания стилей будут доступны всему приложению. Рассмотрим пример:

```
<Application xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
 xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
 x:Class="SilverlightApplication1.App">
 <Application.Resources>
  <Style TargetType="TextBlock" x:Key="TextBlockStyle">
   <Setter Property="FontFamily" Value="Comic Sans Ms"></Setter>
   <Setter Property="Text" Value="Click!"></Setter>
   <Setter Property="Foreground" Value="MediumBlue"></Setter>
   <Setter Property="FontSize" Value="20"></Setter>
  </Style>

 </Application.Resources>
</Application>
```

Применяя этот код, можно задавать стиль любого *TextBlock* в приложении. Как описывалось выше, любое из свойств *Style* можно переопределить, задавая его для конкретного элемента управления.

Шаблоны

В предыдущем разделе была создана кнопка с изображением, используя тот факт, что *Button* является элементом управления *с содержимым*. Это позволило создать не просто кнопку с надписью, а включить в нее XAML, описывающий *StackPanel* с изображением (*Image*) и текстовым блоком (*TextBlock*). Затем вы научились с помощью стилей эффективно применять единообразные настройки к свойствам нескольких элементов *TextBlock*.

Следующий логический шаг – определение стиля сразу *всего* элемента управления: *Button* и образующих его элементов управления, т.е. *StackPanel, Image* и *TextBlock*. Вот где шаблоны становятся основными средствами настройки внешнего вида ваших Silverlight-приложений!

Шаблоны действуют абсолютно аналогично стилям: размещаются в разделе *<Resources>* и описываются с помощью тега *Setter*. Чтобы создать *Template* (Шаблон), задается свойство *Template* и затем с помощью *<Setter.Value>* определяется *ControlTemplate* для заданного элемента управления:

Разобраться в этом поможет код[1]:

```
<Canvas.Resources>
   <Style TargetType="TextBlock" x:Key="TextBlockStyle">
    <Setter Property="FontFamily" Value="Comic Sans Ms"></Setter>
    <Setter Property="Text" Value="Click!"></Setter>
    <Setter Property="Foreground" Value="MediumBlue"></Setter>
    <Setter Property="FontSize" Value="20"></Setter>
   </Style>
   <Style x:Key="ImageButton" TargetType="Button">
    <Setter Property="Template">
     <Setter.Value>
      <ControlTemplate TargetType="Button">
       <Button>
        <Button.Content>
         <StackPanel Orientation="Vertical">
          <Image Source="icon.jpg" Height="48" Width="48"></Image>
          <TextBlock Style="{StaticResource TextBlockStyle}"
                  Foreground="Black">
          </TextBlock>
         </StackPanel>
        </Button.Content>
       </Button>
      </ControlTemplate>
     </Setter.Value>
    </Setter>
   </Style>
 </Canvas.Resources>
```

Здесь создается стиль *ImageButton*, в котором используется Setter для свойства *Template*. В нем содержится *ControlTemplate*, описывающий *Button*, включающую *StackPanel* с *Image* и *TextBlock*.

Обратите внимание, что в этом *TextBlock* также может использоваться ссылка на стиль, потому что для элемента управления *TextBlock*, используемого в шаблоне, задан созданный ранее стиль *TextBlockStyle*.

Теперь XAML для объявления кнопок стал еще проще:

```
 <Button x:Name="btn1" Style="{StaticResource ImageButton}"
       Canvas.Top="20" Canvas.Left="20"></Button>

 <Button x:Name="btn2" Style="{StaticResource ImageButton}"
       Canvas.Top="20" Canvas.Left="120"></Button>

 <Button x:Name="btn3" Style="{StaticResource ImageButton}"
       Canvas.Top="20" Canvas.Left="220"></Button>
```

Для каждой кнопки в качестве значения *Style* задан ресурс *ImageButton*, который, в свою очередь, определяет шаблон для всего элемента управления. На рис. 13-4 показан результат выполнения этого XAML.

[1] Обратите внимание, что стили размещены в ресурсах Canvas, значит, он будет областью их видимости, а не весь UserControl (*прим. редактора*).

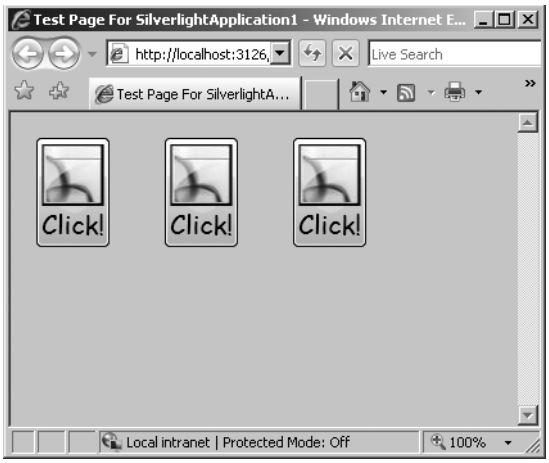

РИС. 13-4 Кнопки, заданные шаблоном.

По-настоящему замечательное свойство шаблонов в том, что в шаблоне не обязательно присутствие конкретного типа элемента управления, шаблон которого создается. Вероятно, звучит немного запутанно, поэтому рассмотрим пример, чтобы разобраться во всем.

Выше в этом разделе был определен шаблон для кнопки с содержимым, которое делает ее простой кнопкой с изображением. Но, чтобы создать шаблон кнопки, элемент управления *Button*, по сути, не нужен. Можно без труда создать кнопки с изображением и текстом, не беря за основу стандартный элемент управления *Button*. Рассмотрим шаблон:

```
<Style x:Key="ImageButton" TargetType="Button">
    <Setter Property="Template">
     <Setter.Value>
       <ControlTemplate TargetType="Button">
         <StackPanel Orientation="Vertical">
           <Image Source="icon.jpg" Height="48" Width="48"></Image>
           <TextBlock Style="{StaticResource TextBlockStyle}"

                   Foreground="Black">

           </TextBlock>
         </StackPanel>
       </ControlTemplate>
     </Setter.Value>
    </Setter>
</Style>
```

Таким образом, мы получили шаблон (заданный с помощью *TargetType*) вообще без элемента управления *Button*, но Silverlight будет интерпретировать его как кнопку, включая все соответствующие свойства и события. Например, если применить этот шаблон (бескнопочный), при выполнении приложения на экране можно будет увидеть примерно следующее (рис. 13-5).

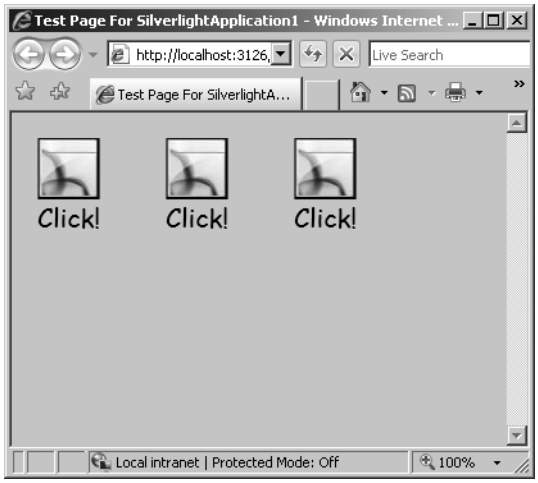

РИС. 13-5 Бескнопочные шаблоны для кнопки.

Убедиться в том, что это настоящие кнопки, можно, попробовав написать код для них. Например, они имеют полную поддержку IntelliSense элемента управления *Button* в Visual Studio и Expression Blend 3. Это можно увидеть на рис. 13-6, где представлено написание кода для события *Click*.

```
public Page()
{
    InitializeComponent();
    btn1.Click+=
}
```

new RoutedEventHandler(btn1_Click); (Press TAB to insert)

РИС. 13-6 Написание кода для кнопок, создаваемых по шаблону.

Поскольку шаблоны – это просто особые стили, их тоже можно описывать в *App.xaml*, чтобы сделать доступными для применения на многих страницах XAML.

Заключение

В данной главе мы познакомились со стилями и шаблонами. Это очень важные возможности Silverlight, потому что они позволяют унифицировать и настраивать внешний вид и поведение всех создаваемых вами элементов управления. Вы научились объявлять стиль и использовать его как статический ресурс (*StaticResource*) элемента управления. Затем было показано, как использовать стиль во всем приложении, задавая его в *App.xaml*.

Также вы увидели, как эта возможность может быть расширена с помощью шаблонов. Шаблоны аналогичны стилям, но позволяют описать стиль всего элемента управления. Был продемонстрирован *ImageButton*, созданный без использования *Button*, но все равно обладающий всеми свойствами, методами и событиями *Button*, благодаря объявлению *ControlTemplate* для него.

В следующей главе будет показано, как можно «оживить» элементы управления с помощью связывания данных и шаблонов.

Глава 14

Связывание данных в Silverlight

В Главе 12, «Создание Silverlight-приложений, взаимодействующих с сервером», были рассмотрены возможности подключения в Microsoft Silverlight и доступа к сервисам для получения данных. Приведены несколько простых примеров связывания данных в формате XML, в которых для создания связываемых объектов использовался LINQ-запрос к XML.

В данной главе обратимся к тому, как самостоятельно создавать объекты и использовать их в простых сценариях со связыванием данных. Объекты являются строительными блоками связывания любых данных в Silverlight, поэтому они помогут разобраться в более сложных сценариях связывания.

Создание объекта данных

В Главе 12 использовался класс данных для представления стоимости акций за период времени. В этой главе расширим этот класс и создадим на его базе связываемый класс, формирующий события, которые можно использовать для перехвата событий связывания данных, таких как изменение значения свойства.

Как обычно, создадим новый проект Silverlight и назовем его CH14_Sample1.

Добавим в него папку Data (Данные) и в нее – новый класс *TimeSeriesBoundData* (Данные за период с поддержкой связывания).

Данный класс реализует интерфейс *INotifyPropertyChanged* (Уведомление об изменении свойства), описанный в *System.ComponentModel*, поэтому не забудьте сослаться на него в начале описания своего класса:

```
using System.ComponentModel;
```

Далее следует обеспечить, чтобы класс реализовывал упомянутый интерфейс *INotifyPropertyChanged*, следующим образом:

```
public class TimeSeriesBoundData : INotifyPropertyChanged
```

Этот класс формирует событие при изменении свойства. Чуть ниже мы рассмотрим, как перехватывать это событие в коде, но прежде всего необходимо объявить это событие в классе:

```
public event PropertyChangedEventHandler PropertyChanged;
```

Если вспомнить Главу 12, класс данных для набора данных имел поля для хранения начальной цены, конечной цены, максимальной цены, минимальной цены, объема, установленной цены закрытия и даты. Рассмотрим здесь, как реализовать это в классе, в котором изменение значения свойства (за счет связывания или другим способом) приводит к формированию события.

Вот как это реализуется для свойства *open*:

```
private double _open;
      public double open
      {
         get { return _open; }
         set
         {
            _open = value;
            OnPropertyChanged("open");
         }
      }
```

Здесь объявляется закрытое поле *_open*, в котором хранится значение. Метод доступа *get* просто возвращает это значение, метод *set* заменяет его переданным значением. Переменная *value*, которую можно видеть в данном фрагменте кода, формируется во время выполнения при любом обновлении свойства, независимо от того изменяется ли оно кодом или через связывание данных.

В этот момент должно формироваться событие изменения свойства, поэтому вызываем *OnPropertyChanged* (Свойство изменилось) и передаем в него строку *open*, указывая, что произошло изменение значения свойства *open*.

Рассмотрим, что делает этот метод:

```
private void OnPropertyChanged(string propertyName)
{
   if (PropertyChanged != null)
   {
      PropertyChangedEventArgs args =
         new PropertyChangedEventArgs(propertyName);
      PropertyChanged(this, args);
   }
}
```

Как видите, все довольно просто. Сначала проводится проверка того, что событие *PropertyChanged* не *null*, т.е. что для события *PropertyChanged* есть хотя бы один обработчик (если его нет, то зачем генерировать событие). Если для события есть обработчики, метод создает *PropertyChangedEventArgs* (помните, что в Microsoft .NET Framework изменения свойств обычно обрабатываются методом, который принимает объект и аргументы в качестве параметров) и инициализирует эти аргументы полученной строкой, которая определена как имя свойства, изменение которого обусловило вызов этого метода.

Вот, практически, и вся функциональность этого класса данных. Для удобства приводим здесь класс полностью со всеми свойствами:

```
using System;
using System.Net;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Documents;
using System.Windows.Ink;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Animation;
using System.Windows.Shapes;
using System.ComponentModel;

namespace CH14_Sample1.Data
{
   public class TimeSeriesBoundData : INotifyPropertyChanged
   {
      public event PropertyChangedEventHandler PropertyChanged;


      private void OnPropertyChanged(string propertyName)
      {
         if (PropertyChanged != null)
         {
            PropertyChangedEventArgs args = new PropertyChangedEventArgs(propertyName);
            PropertyChanged(this, args);
         }
      }

      private DateTime _date;
      public DateTime date
      {
         get { return _date; }
         set
         {
```

```
            _date = value;
            OnPropertyChanged("date");
        }
    }

    private double _open;
    public double open
    {
        get { return _open; }
        set
        {
            _open = value;
            OnPropertyChanged("open");
        }
    }

    private double _close;
    public double close
    {
        get { return _close; }
        set
        {
            _close = value;
            OnPropertyChanged("close");
        }
    }

    private double _high;
    public double high
    {
        get { return _high; }
        set
        {
            _high = value;
            OnPropertyChanged("high");
        }
    }

    private double _low;
    public double low
    {
        get { return _low; }
        set
        {
            _low = value;
            OnPropertyChanged("low");
        }
    }

    private double _volume;
    public double volume
    {
        get { return _volume; }
        set
        {
            _volume = value;
            OnPropertyChanged("volume");
        }
    }

    private double _adjClose;
    public double adjClose
    {
        get { return _adjClose; }
        set
        {
            _adjClose = value;
            OnPropertyChanged("adjClose");
        }
    }
  }
}
```

Теперь, имея класс, который может быть связан и формирует события, рассмотрим простой сценарий связывания этого класса с элементами пользовательского интерфейса в XAML.

Связывание с объектом данных

В созданном ранее проекте имеется класс MainPage.xaml, содержащий *UserControl*. Помните, вы создали связываемый класс в папке Data? Если внимательней посмотреть на этот класс, можно увидеть, что он был помещен в подпространство имен *Data* основного пространства имен вашего приложения.

Итак, если пространство имен приложения – *CH14_Sample1*, пространство имен связываемого класса будет *CH14_Sample1.Data*.

Чтобы использовать его в пользовательском интерфейсе, не забудьте создать ссылку на него в XAML. Объявите пространство имен в начале XAML-кода.

Это должно выглядеть следующим образом:

```
xmlns:data ="clr-namespace:CH14_Sample1.Data"
```

Вот полное описание заголовка *<UserControl>*, включая данное объявление:

```
<UserControl x:Class="CH14_Sample1.MainPage"
   xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
   xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
   xmlns:data ="clr-namespace:CH14_Sample1.Data"
   Width="640" Height="480">
```

Для простого связывания данных можно добавить экземпляр класса *TimeSeriesBoundData* в *UserControl* как ресурс.

Чтобы добавить ресурс в *UserControl*, используем *<UserControl.Resources>* следующим образом:

```
<UserControl.Resources>
</UserControl.Resources>
```

Добавив ссылку на пространство имен *CH14_Sample1.Data* в заголовки *UserControl*, вы определили префикс *data* для классов данного пространства имен. Для создания экземпляра *TimeSeriesBoundData* под именем *TSD* используется такой XAML:

```
<data:TimeSeriesBoundData x:Key="TSD"></data:TimeSeriesBoundData>
```

Итак, вот как выглядит ваш XAML-код:

```
   <UserControl.Resources>
      <data:TimeSeriesBoundData x:Key="TSD"></data:TimeSeriesBoundData>
   </UserControl.Resources>
```

Теперь, когда экземпляр *TimeSeriesBoundData* хранится на странице как ресурс, пришла пора выполнить связывание с ним. Связывание реализуется с помощью *TextBox*, который отображает значение одного из свойств *TimeSeriesBoundData*, а также позволяет *изменять* это значение. Кроме того, добавим *TextBlock*, отображающий значение свойства *open*.

Сначала рассмотрим XAML-код для *TextBox*:

```
<TextBox Canvas.Top="0" x:Name="txtInput"
    Text="{Binding Source={StaticResource TSD}, Path=open, Mode=TwoWay}">
</TextBox>
```

Вероятно, вам уже знакомо свойство *Text* элемента управления *TextBox*. Свойство *Text* позволяет задавать текст, используемый для инициализации *TextBox*. В данном случае имеем странное значение в фигурных скобках. Это значение называется *Binding* (Привязка). Оно указывает, что выполняется

привязка к ресурсу *TSD*, описанном в разделе *UserControl.Resources*, что определяющим свойством (определяется с помощью *Path*,т.е. пути к необходимому свойству) является *open*, и, наконец, что используется режим связывания *TwoWay* (Двухсторонний), т.е. что можно как читать, *так и* записывать значения обратно. Таким образом, данная привязка обеспечивает возможность отображать значения связанных данных и также записывать значения в них.

Тот же принцип применяется и к *TextBlock*, но поскольку *TextBlock* только отображает данные и не предоставляет возможности их *ввода*, привязка должна быть односторонней. Рассмотрим XAML:

```
<TextBlock Canvas.Top="20" x:Name="txtView"
   Text="{Binding Source={StaticResource TSD}, Path=open, Mode=OneWay}">
</TextBlock>
```

Как видите, значение *Mode* (Режим) в данном случае *OneWay* (Односторонний). Во всем остальном данное описание идентично описанию *TextBox*.

Рассмотрим полный XAML этого приложения:

```
<UserControl x:Class="CH14_Sample1.MainPage"
   xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
   xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
   xmlns:data ="clr-namespace:CH14_Sample1.Data"
   Width="640" Height="480">
   <UserControl.Resources>
      <data:TimeSeriesBoundData x:Key="TSD"></data:TimeSeriesBoundData>
   </UserControl.Resources>
   <Canvas x:Name="LayoutRoot" Background="White">
      <TextBox Canvas.Top="0" x:Name="txtInput"
         Text="{Binding Source={StaticResource TSD}, Path=open, Mode=TwoWay}">
      </TextBox>
      <TextBlock Canvas.Top="20" x:Name="txtView"
         Text="{Binding Source={StaticResource TSD}, Path=open, Mode=OneWay}">
      </TextBlock>
   </Canvas>
</UserControl>
```

При выполнении этого приложения вы видите простой UI с *TextBox* и *TextBlock*, как показано на рис. 14-1.

РИС. 14-1 Базовый пользовательский интерфейс со связыванием данных.

Обратите внимание, что и в *TextBlock*, и в *TextBox* отображается значение 0. Причина в том, что данные не были инициализированы, а значение *open* типа double, т.е. по умолчанию равно 0.

Также заметьте, что ввод значений в *TextBox* ни к чему не приводит. Это объясняется тем, что связывание происходит при изменении *TextBox*, а событие изменения имеет место, когда *TextBox* теряет фокус в результате нажатия клавиши Tab или перехода пользователя на другую часть UI посредством мыши. В данном UI имеется только *TextBlock*, который не может получить фокус. Поэтому перейти с *TextBox* и увидеть изменение *TextBlock* в данном случае можно только с помощью клавиши Tab.

Результаты связывания данных представлены на рис. 14-2.

РИС. 14-2 Связывание данных в действии.

Итак, вы создали свой первый связываемый класс и увидели, как выполнять связывание с помощью связывания данных XAML.

Это связывание данных обеспечивает прекрасные результаты для чисел, как видно на рис. 14-2, но что произойдет, если вместо *123* ввести *123ABC*? Как мы помним, *open* – значение типа double, а *123ABC* не double, т.е. недопустимое, поэтому при вводе такого значения ничего не произойдет. Следующий раздел посвящен подробному рассмотрению этого вопроса и рассказывает об использовании специального класса для преобразования данных при связывании, что обеспечивает корректное связывание объектов.

Преобразования при связывании

Если выполнить предыдущий пример этой главы и попытаться ввести в *TextBox* недопустимое значение (такое как *123ABC*), связывания не происходит, потому что возникает ошибка. Ваш код не обеспечивает перехвата и обработки этой ошибки, поэтому на данный момент просто не будет ничего происходить.

Ошибку можно увидеть в панели Output (Вывод) в Microsoft Visual Studio. Она должна выглядеть примерно так:

```
System.Windows.Data Error: ConvertBack cannot convert value '123abc' (type 'System.String'). BindingExpression:
Path='open' DataItem='CH14_Sample1.Data.TimeSeriesBoundData' (HashCode=35912612); target element is
'System.Windows.Controls.TextBox' (Name='txtInput'); target property is 'Text' (type 'System.String')..
System.FormatException: Input string was not in a correct format.
   at System.Number.StringToNumber(String str, NumberStyles options, NumberBuffer& number, NumberFormatInfo info,
Boolean parseDecimal)
   at System.Number.ParseDouble(String value, NumberStyles options, NumberFormatInfo numfmt)
   at System.Double.Parse(String s, NumberStyles style, NumberFormatInfo info)
   at System.Double.Parse(String s, NumberStyles style, IFormatProvider provider)
   at System.Convert.ToDouble(String value, IFormatProvider provider)
   at System.String.System.IConvertible.ToDouble(IFormatProvider provider)
   at System.Convert.ChangeType(Object value, Type conversionType, IFormatProvider provider)
   at MS.Internal.Data.SystemConvertConverter.Convert(Object o, Type type, Object parameter, CultureInfo culture)
   at MS.Internal.Data.DynamicValueConverter.Convert(Object value, Type targetType, Object parameter, CultureInfo culture)
   at System.Windows.Data.BindingExpression.UpdateValue().
```

Малопонятно. Так как же обработать эту ошибку? Можно либо написать огромное количество кода для обработки каждой возможной ошибки связывания, либо можно применить функциональность *Converter* (Конвертер), которая доступна как часть синтаксиса связывания данных. *Converter* определяет класс, обеспечивающий очистку данных как часть связывания.

Добавим в папку Data новый класс *ToDoubleConverter*. Он реализует интерфейс *IValueConverter* (Конвертер значений) из пространства имен *System.Windows.Data*. Добавим следующее описание в начало класса:

```
using System.Windows.Data;
```

Чтобы реализовать этот класс, необходимо изменить его объявление следующим образом:

```
public class ToDoubleConverter : IValueConverter
```

Данный интерфейс требует реализации методов *Convert* (Преобразовать) и *ConvertBack* (Выполнить обратное преобразование). Первый используется при чтении данных для привязки, второй – при записи.

Итак, в обоих случаях необходимо гарантированно обеспечить значение типа double. Обратите внимание, в данном примере это не так важно для *Convert*, потому что ошибка возникает при вводе строки в *TextBlock*, что обусловливает вызов *ConvertBack*.

```
public object Convert(object value, Type targetType,
   object parameter, System.Globalization.CultureInfo culture)
      {
         double nReturn = 0.0;
         double.TryParse(value.ToString(), out nReturn);
         return nReturn;
      }

      public object ConvertBack(object value, Type targetType,
   object parameter, System.Globalization.CultureInfo culture)
      {
         double nReturn = 0.0;
         double.TryParse(value.ToString(), out nReturn);
         return nReturn;
      }
```

Преобразование довольно простое – создается значение типа double, содержащее 0, и затем с помощью NET-метода *TryParse* (Попытаться выполнить синтаксический разбор) выполняется проверка возможности представления строки как значения double. Если это возможно, выполняется преобразование; если нет, значение остается равным 0.

Рассмотрим полный код класса *ToDoubleConverter* (Конвертер в тип Double):

```
using System;
using System.Net;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Documents;
using System.Windows.Ink;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Animation;
using System.Windows.Shapes;
using System.Windows.Data;

namespace CH14_Sample1.Data
{
   public class ToDoubleConverter : IValueConverter
   {

      #region IValueConverter Members

      public object Convert(object value, Type targetType, object parameter, System.Globalization.CultureInfo culture)
      {
         double nReturn = 0.0;
         double.TryParse(value.ToString(), out nReturn);
         return nReturn;
      }



      public object ConvertBack(object value, Type targetType, object parameter, System.Globalization.CultureInfo culture)
      {
         double nReturn = 0.0;
         double.TryParse(value.ToString(), out nReturn);
         return nReturn;
      }

      #endregion
```

```
    }
}
```

Теперь следует откомпилировать ваше приложение и убедиться в отсутствии ошибок. Это обеспечит работу функциональности автозаполнения Visual Studio на следующем этапе, при добавлении в приложение экземпляра класса *Converter*.

Вернитесь в раздел *<UserControl.Resources>* XAML-кода и добавьте экземпляр *ToDoubleConverter* следующим образом:

```
<data:ToDoubleConverter x:Key="dblConverter"></data:ToDoubleConverter>
```

Итак, теперь раздел *Resources* выглядит так:

```
<UserControl.Resources>
    <data:TimeSeriesBoundData x:Key="TSD"></data:TimeSeriesBoundData>
    <data:ToDoubleConverter x:Key="dblConverter"></data:ToDoubleConverter>
</UserControl.Resources>
```

Как видите, экземпляр класса *Converter* назван *dblConverter*.

Последний шаг – реализовать конвертер как часть привязки. Преобразование имеет место при вводе недопустимого значения (такого как *ABC123*) в *TextBox*, поэтому необходимо определить использование экземпляра конвертера как части привязки.

Вот новое описание *TextBox*:

```
<TextBox Canvas.Top="0" x:Name="txtInput"
    Text="{Binding Source={StaticResource TSD}, Path=open, Mode=TwoWay,
        Converter={StaticResource dblConverter}}">
</TextBox>
```

Теперь при выполнении приложения и вводе недопустимого значения в *TextBox* оно будет преобразовано в 0, и связывание будет выполняться корректно.

Перехват событий изменений свойств

Ранее при создании связываемого класса для набора данных обеспечивалось возникновение события при изменение одного из его свойств. Эти события являются стандартными, поэтому их достаточно прсто перехватывать. Также в приложение был добавлен класс *TimeSeriesBoundData* как ресурс под именем *TSD*.

Посмотрим на конструктор Silverlight-приложения. XAML-файл называется MainPage.xaml по умолчанию, поэтому конструктор называется *MainPage()*.

Добавьте в файл ссылку на пространство имен

```
using CH14_Sample1.Data;
```

В конструкторе после вызова *InitializeComponent()* (Инициализировать компонент) добавьте следующий код, чтобы получить ссылку на экземпляр класса, который является ресурсом:

```
TimeSeriesBoundData theData = this.Resources["TSD"] as TimeSeriesBoundData;
```

Имея эту ссылку, можно задать обработчик события *PropertyChanged*. Если в Visual Studio ввести **theData** и поставить точку (**.**), открывается контекстное меню с доступными свойствами, методами и событиями. Просмотрите предлагаемый список и увидите в нем событие *PropertyChanged*. Выберите его и нажмите клавишу Tab. Введите **+=** и увидите подсказку с предлагаемым кодом. Нажмите Tab, чтобы ввести этот код, и затем нажмите Tab снова, чтобы сгенерировать заглушку для обработчика события.

Заглушка выглядит следующим образом:

```
void theData_PropertyChanged(object sender, System.ComponentModel.PropertyChangedEventArgs e)
{}
```

Теперь при любом изменении значения любого из свойств будет вызываться этот обработчик события.

В объекте аргументов *e* имеется свойство *PropertyName* (Имя свойства), описанное в этой главе ранее. *PropertyName* – это строка, содержащая имя измененного свойства.

Прочитать его не составляет труда:

```
string theProperty = e.PropertyName;
```

Имея имя измененного свойства, можно выполнять любые дополнительные операции.

Если добавить точку останова в приведенной выше строке кода и запустить приложение, можно увидеть, что *e.PropertyName* возвращает *open*.

Описание привязок

В данной главе представлен ряд примеров привязок, описываемых с помощью фигурных скобок и списка свойств, определяющих поведение привязки.

Рассмотрим эти свойства, различные значения, поддерживаемые ими, и их использование:

- ***Source*** *Source* определяет, где происходит связывание. Это может быть ресурс в XAML, в этом случае он задается как *StaticResource* (Статический ресурс), или другой связываемый источник, например *IEnumerable*, как обсуждалось в Главе 12.
- ***Path*** *Path* – это идентификатор свойства, которое связывается с источником.
- ***Mode*** *Mode* определяет режим связывания. Может принимать значения *OneWay*, что определяет связывание только с возможностью чтения; *TwoWay*, что означает связывание с возможностью как чтения, так и записи; или *OneTime*, что определяет одноразовое связывание, выполняемое лишь при первом формировании отображения элемента управления.
- ***Converter*** *Converter* используется, если необходимо преобразование данных при связывании. Пример этого представлен в данной главе. *Converter* включает имя экземпляра класса *Converter*. Также могут использоваться *ConverterParameter* (Параметр конвертера) и *ConverterCulture* (Региональные настройки конвертера) для более тонкой обработки. Эти вопросы выходят за рамки обсуждения данной книги.

Заключение

В данной главе рассмотрены основы связывания данных в Silverlight, созданы связываемый класс и простое приложение, обеспечивающее как одностороннее, так и двухстороннее связывание. Также дан пример обработки ошибок при связывании данных с помощью *Converter* и приведены элементы, используемые XAML для описания связывания. Данная глава предлагает лишь основные сведения, которые позволяют понять принципы связывания данных в таких сложных элементах управления как *DataGrid*, и на базе которых можно создавать собственные приложения со связыванием данных.

Глава 15

Использование динамических языков в Silverlight

Microsoft Silverlight поддерживает *динамические* языки программирования. Динамическими называют языки программирования, которые реализуют многие функции во *время выполнения*, в то время как в других языках, например C#, это происходит во *время компиляции*. Динамическое поведение включает расширение объектов и описаний, изменение системы типов и многое другое. Динамические языки обеспечивают простой подход к разработке через цикл «выполнение-вычисление-вывод» с применением метода проб и ошибок в противоположность традиционной модели «написание кода-компиляция-тестирование-перекомпиляция».

Silverlight поддерживает три самых популярных динамических языка: Ruby, Python и Dynamic JavaScript. Данная книга написана для спецификации бета-версии Silverlight 3, в которую не включен шаблон проекта Microsoft Visual Studio 2008, но пакет для разработчика ПО (SDK) предоставляет инструмент, Chiron.exe, для работы с динамическими языками программирования.

Поддержка динамических языков является инициативой Microsoft с открытым исходным кодом и доступна на Веб-сайте CodePlex по адресу *http://www.codeplex.com/sdlsdk*. В данной главе будет рассмотрено построение базовых приложений для Silverlight с использованием динамических языков. В данной главе не ставится задача изучения самих языков программирования и используются очень простые программы, которые будут понятны даже людям, не являющимся экспертами в каком-то конкретном языке.

Эта глава является только лишь введением в динамические языки и их использование в Silverlight. Большая ее часть посвящена рассмотрению Visual Studio и возможностей ее применения для создания приложений на IronPython, IronRuby и управляемом JavaScript в Silverlight, а также интегрированию Chiron.exe в Visual Studio. А напоследок вы совершите тур по знаменитому примеру Silverlight с часами и узнаете, как он реализован на Ruby.

Ваше первое Silverlight-приложение на IronPython

С инструментом Chiron проще всего работать, если настроить специальную структуру каталогов для своего приложения. В данном разделе будет поэтапно рассмотрено, как это сделать.

Прежде всего, необходимо создать *пустой* Веб-сайт. Для этого запустим Visual Studio и выберем в меню File пункт New Web Site (Новый Веб-сайт). Опция Empty Web Site (Пустой Веб-сайт) предлагается в диалоговом окне New Web Site, как показано на рис. 15-1.

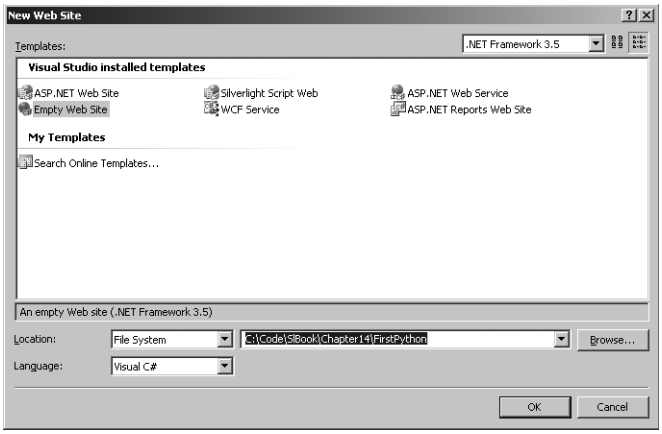

РИС. 15-1 Диалоговое окно New Web Site.

Это обеспечит создание пустого каталога, который для Visual Studio будет являться корневым каталогом Веб-сайта. Для создания сайта будет использоваться Chiron, но вы увидите, что создавать страницы, код и т.д. проще в Visual Studio.

В Solution щелкните правой кнопкой на папке проекта и выберите Add New Item. Откроется диалоговое окно Add New Item, которое может использоваться для создания новой HTML-страницы. Выберите опцию HTML Page (HTML-страница) и присвойте ей имя Default.html (рис. 15-2).

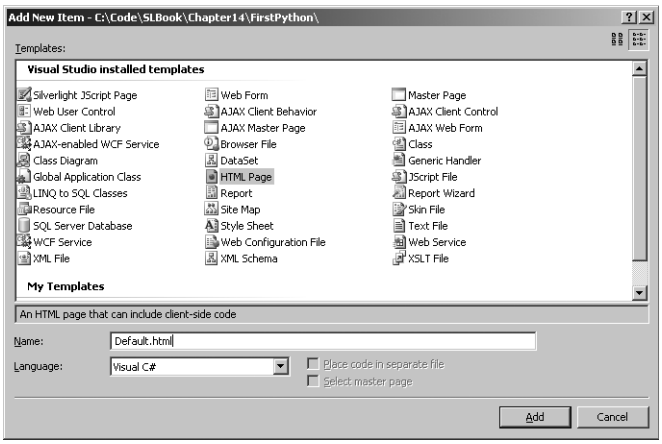

РИС. 15-2 Добавление нового элемента на сайт.

Это обеспечит вам базовую HTML-страницу. Ее необходимо отредактировать и превратить в страницу, на которой будет располагаться приложение Silverlight. Рассмотрим код такой страницы:

```
<html xmlns="http://www.w3.org/1999/xhtml" >
<head>
  <title>Dynamic Silverlight Test Page </title>
  <style type="text/css">
    html, body {
      height: 100%;
      overflow: auto;
    }
    body {
      padding: 0;
      margin: 0;
    }
    #silverlightControlHost {
      height: 100%;
    }
  </style>
</head>
<body>
  <div id="silverlightControlHost">
   <object data="data:application/x-silverlight-2,"
    type="application/x-silverlight-2"
    width="100%" height="100%">
```

```
    <param name="source" value="app.xap"/>
    <param name="background" value="white" />
    <param name="windowless" value="true" />
   </object>
   <iframe style='visibility:hidden;height:0;width:0;border:0px'></iframe>
  </div>
</body>

</html>
```

Для размещения Silverlight на странице используется тег *<object>*. Это очень упрощенный пример, и он не включает код для установки Silverlight, если его нет в системе. Более подробно об этом рассказывается в Главе 6, «Элемент управления браузера Silverlight».

Обратите внимание на параметр *source*, которому задано значение *app.xap*. Chiron создаст его автоматически, как мы увидим через мгновение. Чтобы *xap*-файлу было присвоено такое имя, ваш файл кода должен называться app. Обратите внимание, что Chiron не компилирует код в XAP-файл, он просто помещает весь сценарий в этот файл, чтобы Silverlight мог загружать его.

Сначала рассмотрим, как создавать приложение с использованием IronPython и файла app.py, а затем реализуем ту же функциональность на IronRuby и Dynamic JavaScript.

Далее создадим каталог app в вашем Веб-сайте. Для него с помощью Add New Item выберем шаблон XML File (XML-файл) и назовем app.xaml.

Вот пример простого XAML, который можно поместить в этот файл:

```
<UserControl x:Class="System.Windows.Controls.UserControl"
   xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
   xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml" >

  <Grid x:Name="layout_root" Background="White">
   <TextBlock x:Name="Message" FontSize="30" Text="Hello World," />
  </Grid>

</UserControl>
```

Здесь описывается объект *UserControl*, включающий *TextBlock* под именем *txtMessage*. Данный фрагмент содержит текст «Hello World,». Обратите внимание на запятую в конце фразы, почему она там находится, станет понятно через мгновение.

Теперь добавим файл кода IronPython. Опять же, щелкните правой кнопкой на папке app Веб-сайта и выберите Add New Item. В диалоговом окне выберите Text File (Текстовый файл) и назовите его app.py (py – расширение, используемое для Python).

На рис. 15-3 представлено, как должно выглядеть ваше решение на данный момент.

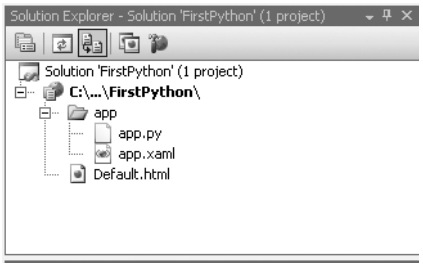

РИС. 15-3 Структура решения.

Теперь можно приступать к редактированию файла кода Python, чтобы сделать что-то более интересное. Рассмотрим код:

```
from System.Windows import Application
from System.Windows.Controls import UserControl
```

```
def handleClick(sender, eventArgs):
   sender.Text = sender.Text + " from Python!"

class App:
   def __init__(self):
      self.scene =
        Application.Current.LoadRootVisual(UserControl(),"app.xaml")

   def start(self):
      self.scene.txtMessage.MouseLeftButtonUp += handleClick

App().start()
```

Если у вас есть опыт разработки приложений Silverlight на C# или Microsoft Visual Basic, синтаксис будет вам знаком. Например, строка

```
from System.Windows import Application
```

эквивалентна строке на C#

```
using System.Windows;
```

и указывает на пространство имен, в котором находится класс *Application*.

Затем определяем Python-класс *App* и задаем объект *scene* для *App*, загружая XAML-файл, описанный ранее. Как часть описания класса задается обработчик события запуска:

```
def start(self):
```

и код, который будет выполняться в этом обработчике. Этот код определяет обработчик события *MouseLeftButtonUp* элемента управления *txtMessage* типа *TextBlock*, который был описан нами ранее. Назовем обработчик *handleClick* (Обработка щелчка).

Сам код этого обработчика события выглядит следующим образом:

```
def handleClick(sender, eventArgs):
   sender.Text = sender.Text + " from Python!"
```

Выполнение этого кода обеспечивает добавление фразы «from Python!» (из Python!) в конец значения свойства *Text* отправителя (которым, в данном случае, является *TextBlock*).

Чтобы выполнить это приложение, используется Chiron. Настроить Visual Studio на запуск Chiron можно, щелкнув правой кнопкой мыши файл проекта в Solution Explorer (заметьте, вы должны использовать файл проекта, а не файл решения; файла проекта обычно является первым дочерним файлом в решении) и выбрав в контекстном меню опцию Property Pages (Страницы свойств).

На Property Pages выберите страницу Start Options (Опции запуска) и на ней опцию Start External Program (Запустить внешнюю программу). Используйте кнопку с многоточием (...) для перехода к каталогу, в котором находится Chiron.exe. Это будет каталог в папке bin Пакета для разработчика ПО с использованием динамических языков программирования (Dynamic Languages SDK) для Silverlight. (SDK полностью под именем AgDLR можно загрузить по адресу *http://www.codeplex.com/sdlsdk*.)

Введите **/b** в поле Command Line Arguments (Аргументы командной строки) и введите местоположение Веб-сайта в поле Working Directory (Рабочий каталог). Все это можно увидеть на рис. 15-4.

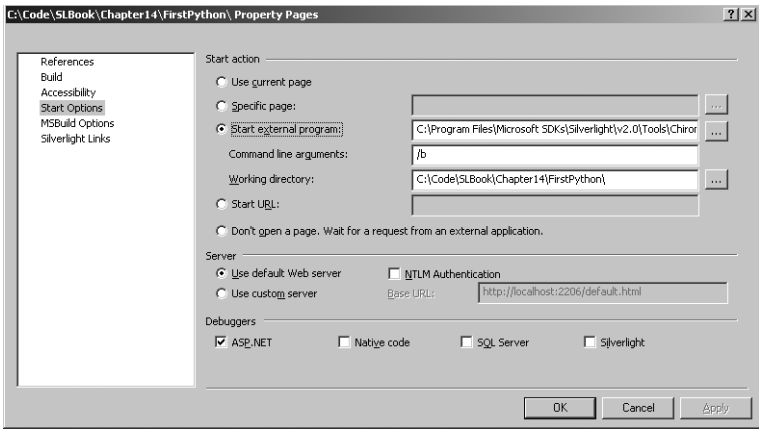

РИС. 15-4 Настройка Visual Studio на использование Chiron.

Теперь можно выполнить наше динамическое приложение Silverlight, нажав F5 в Visual Studio. Запустится Chiron, подтверждение этому вы увидите в окне командной строки, которое представлено на рис. 15-5.

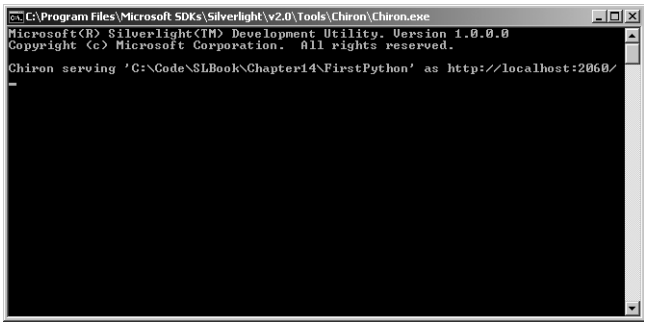

РИС. 15-5 Visual Studio выполняет Chiron.

Здесь сообщается, что доступ к приложению осуществляется через *http://localhost:2060*. Visual Studio также запустит браузер с переходом по этому адресу, что обеспечит вывод на экран списка файлов Веб-каталога, как показано на рис. 15-6.

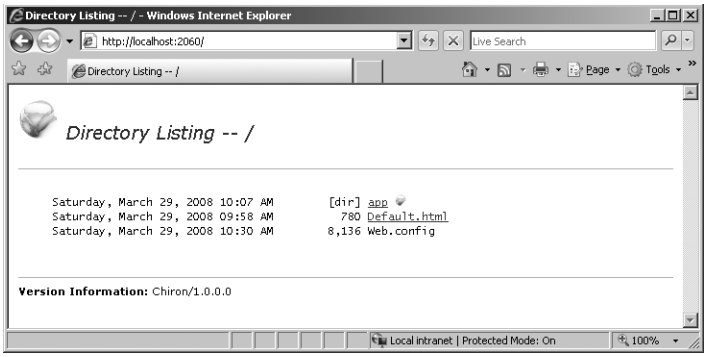

РИС. 15-6 Список файлов каталога Chiron.

Выберите Default.html, чтобы увидеть свое динамическое приложение Silverlight. В браузере будет выведено сообщение «Hello World,», как показано на рис. 15-7.

РИС. 15-7 Ваше первое Silverlight-приложение на Python.

Теперь, если щелкнуть текст «Hello World,», появится дополнительное сообщение «from Python!» (рис. 15-8).

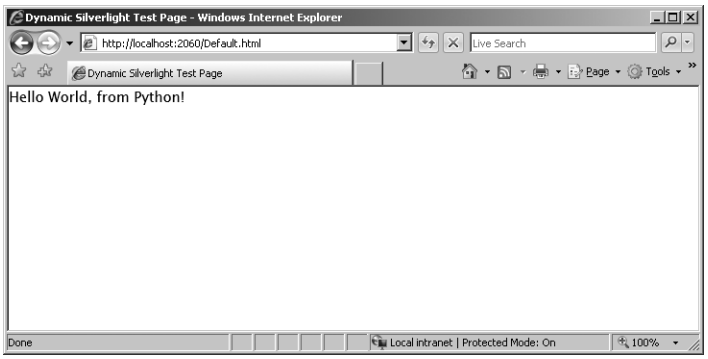

РИС. 15-8 Динамическое приложение в действии.

Это предельно простое приложение, но оно демонстрирует работу Dynamic Language Runtime (Динамическая среда выполнения) в Silverlight и то, как можно настроить Visual Studio для использования динамических языков программирования. В следующем разделе рассматривается, как можно реализовать это же приложение на IronRuby и управляемом JavaScript.

Использование Ruby и JavaScript

В предыдущем разделе мы поэтапно рассмотрели процесс настройки Visual Studio для создания динамических приложений на IronPython. Dynamic Language Runtime также поддерживает языки Ruby и Dynamic JavaScript, поэтому рассмотрим, как можно создать такое же приложение с их использованием.

Использование Ruby

Начнем процесс с создания структуры проекта, включая HTML-файл и XAML-файл, аналогичной той, что использовалась для IronPython в предыдущем разделе. Но в данном случае, в папке app вместо файла app.py для кода на Ruby создается файл app.rb. Также не забудьте выполнить все настройки для запуска Chiron, так же как это делалось в предыдущем примере. (Вернитесь к рис. 15-4, на котором представлены все необходимые настройки).

Вставьте в app.rb код на IronRuby[1]:

```
include System::Windows
include System::Windows::Controls

def handleClick(sender, eventArgs)
  sender.text = "Hello World, from Ruby!"
end
```

[1]Строки с "app.xaml") и { |sender, args| handleClick(sender, args) } не должны переноситься (*прим. редактора*).

```
class App
  def initialize
       @scene = Application.Current.LoadRootVisual(UserControl.new(),
                                        "app.xaml")
  end

  def start
     @scene.find_name('txtMessage').mouse_left_button_up
     { |sender, args| handleClick(sender, args) }
  end
end
App.new.start
```

Как видите, все очень похоже на пример с IronPython. Включаются пространства имен *System.Windows* и *System.Windows.Controls*. Затем описывается обработчик события *handleClick* для изменения текста по щелчку.

После этого объявляется класс *App*. Конструктор класса загружает XAML и использует его для инициализации класса. Событие *Start* связывает обработчик события с *TextBlock txtMessage*, описанным в XAML.

Теперь при выполнении приложения мы получаем такие же результаты: *TextBlock* отображает «Hello World,» и затем, когда пользователь щелкает текст, к этому тексту добавляется «from Ruby!».

Использование Dynamic JavaScript

Чтобы использовать JavaScript, выполните ту же процедуру, что в предыдущих двух примерах; т.е. создайте Веб-сайт в Visual Studio и добавьте в него файлы HTML и XAML. Не забудьте также настроить проект на использование Chiron. Как это сделать, показано на рис. 15-4.

На этот раз, вместо файлов app.py или app.rb в каталог app добавляется файл app.jsx. Редактируйте содержимое файла app.jsx, добавляя в него следующий JavaScript-код:

```
Import("System.Windows.Application")
Import("System.Windows.Controls.UserControl")

function handleClick(sender, eventArgs) {
   sender.Text = sender.Text + " from Dynamic JavaScript";
}

function App() {
   this.scene =
     Application.Current.LoadRootVisual(new UserControl(),
                                  "app.xaml")
}

App.prototype.start = function() {
   this.scene.txtMessage.MouseLeftButtonUp += handleClick
}


app = new App
app.start()
```

Как видите, код по-прежнему предельно понятен. Управляемый JavaScript использует выражение *Import* для добавления ссылок на классы *SystemWindows.Application* и *System.Windows.Controls.UserControl*. В JavaScript само приложение является функцией, в которую добавляются свойства, поэтому *scene* является членом функции *App* и описывается путем загрузки XAML. Событие *MouseLeftButtonUp* обрабатывается функцией *handleClick*. Это определяется в функции *start* приложения.

Как и в предыдущих примерах, *handleClick* добавляет определенный текст, в данном случае, это «from Dynamic JavaScript» (из Dynamic JavaScript), в строку в *txtMessage TextBlock*, описанного в XAML.

Результат абсолютно аналогичен предыдущим двум примерам. Приложение отображает сообщение «Hello World,» и выводит дополнительный текст, когда пользователь щелкает сообщение.

Более сложный пример

Первым приложением Silverlight, которое увидело свет (еще когда Silverlight назывался WPF/E), было приложение с часами. Как оно выглядит, представлено на рис. 15-9.

Это замечательный пример Silverlight-приложения, потому что он демонстрирует многие принципы программирования в Silverlight. Нет необходимости программировать сами часы, все делает анимация Silverlight. Нам необходимо только установить стрелки часов в исходное положение, соответствующее системному времени.

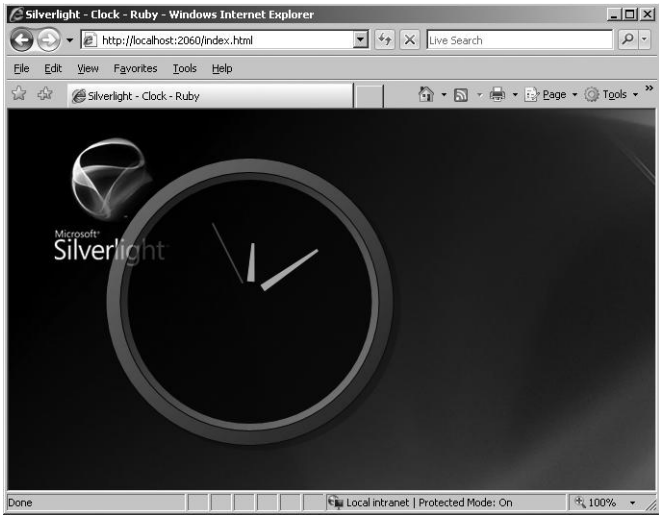

РИС. 15-9 Часы на Silverlight.

Сначала рассмотрим XAML для часов. Обратите внимание, что это лишь фрагмент XAML, представляющий три стрелки часов и их анимацию:

```
<Canvas x:Class="System.Windows.Controls.Canvas"
   xmlns="http://schemas.microsoft.com/client/2007"
   xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
   Opacity="0" x:Name="parentCanvas">

  <Canvas.Triggers>
   <EventTrigger RoutedEvent="Canvas.Loaded">
    <EventTrigger.Actions>
     <BeginStoryboard>
      <Storyboard>
       <DoubleAnimation x:Name="hour_animation"
          Storyboard.TargetName="hourHandTransform"
          Storyboard.TargetProperty="Angle"
          From="180" To="540"
          Duration="12:0:0"
          RepeatBehavior="Forever"/>
       <DoubleAnimation x:Name="minute_animation"
          Storyboard.TargetName="minuteHandTransform"
          Storyboard.TargetProperty="Angle"
          From="180" To="540"
          Duration="1:0:0"
          RepeatBehavior="Forever"/>
       <DoubleAnimation x:Name="second_animation"
          Storyboard.TargetName="secondHandTransform"
          Storyboard.TargetProperty="Angle"
          From="180" To="540"
          Duration="0:1:0"
```

```
            RepeatBehavior="Forever"/>
        </Storyboard>
      </BeginStoryboard>
    </EventTrigger.Actions>
  </EventTrigger>
 </Canvas.Triggers>

 <!—Часовая стрелка -->
 <Path Data="M -4, 16 l 3 40 3 0 2 -40 z" Fill="white">
  <Path.RenderTransform>
    <TransformGroup>
      <RotateTransform x:Name="hourHandTransform"
            Angle="180"/>
      <TranslateTransform X="150.5" Y="145"/>
    </TransformGroup>
  </Path.RenderTransform>
 </Path>

 <!—Минутная стрелка -->
 <Path Data="M -4, 16 l 3 70 3 0 2 -70 z" Fill="white">
  <Path.RenderTransform>
    <TransformGroup>
      <RotateTransform x:Name="minuteHandTransform"
            Angle="180"/>
      <TranslateTransform X="150.5" Y="145"/>
    </TransformGroup>
  </Path.RenderTransform>
 </Path>

 <!—Секундная стрелка -->
 <Path Data="M -1, 16 l 0 70 2 0 0 -70 z" Fill="red">
  <Path.RenderTransform>
    <TransformGroup>
      <RotateTransform x:Name="secondHandTransform"
            Angle="180"/>
      <TranslateTransform X="150.5" Y="145"/>
    </TransformGroup>
  </Path.RenderTransform>
 </Path>

</Canvas>
```

XAML начинается с размещения трех анимаций в соответствующих элементах *Storyboard*. Стрелки часов можно рассматривать как вращающиеся прямые, поэтому для их анимации используется тип *DoubleAnimation*, обеспечивающий изменение угла поворота. Подробнее о реализации этого в XAML рассказывает Глава 5, «Трансформация и анимация в XAML».

Следует обратить внимание на имена анимаций: *hour_animation* (анимация часов), *minute_animation* (анимация минут) и *second_animation* (анимация секунд). Помните об этом при рассмотрении кода. В качестве цели каждой анимации задана именованная трансформация: *hourHandTransform* (Трансформация часовой стрелки) реализует отсчет часов, *minuteHandTransform* (Трансформация минутной стрелки) реализует отсчет минут и *secondHandTransform* (Трансформация секундной стрелки) отвечает за перемещение секундной стрелки. Эти трансформации можно будет увидеть ниже в XAML. Каждая стрелка реализована как объект *Path*, и с каждым из этих контуров ассоциирована одна из упомянутых выше трансформаций.

Итак, анимации запускаются в *Canvas.Loaded* и повторяются постоянно, обеспечивая перемещение стрелок по циферблату часов на 360 градусов за заданный промежуток времени. Но как инициализировать их, чтобы они представляли текущее время? Вот для этого понадобиться написать код.

Реализация часов выполняется в двух модулях кода на Ruby. Первый – вспомогательный класс, созданный группой Silverlight Dynamic Languages (Динамические языки Silverlight) для использования в Ruby-приложениях. Рассмотрим его код:

```
include System::Windows
include System::Windows::Controls
include System::Windows::Media

class SilverlightApplication
  def application
    Application.current
  end

  def self.use_xaml(options = {})
    options = {:type => UserControl, :name => "app"}.merge(options)
      Application.current.load_root_visual(options[:type].new, "#{options[:name]}.xaml")
  end

  def root
    application.root_visual
  end

  def method_missing(m)
    root.send(m)
  end
end

class FrameworkElement
  def method_missing(m)
    find_name(m.to_s.to_clr_string)
  end
end
```

Здесь определяется ряд вспомогательных функций и объектов, включая:

- ***application (приложение)*** Указывает на текущее приложение (просто обозначается *Application.current*)
- ***root (корень)*** Указывает на корневой визуальный элемент, а именно, элемент верхнего уровня в XAML
- ***method_missing (поиск метода по имени)*** Обрабатывает трансляцию имени метода в строковый формат *clr*, что помогает при написании кода
- ***use_xaml (использовать xaml)*** Управляет загрузкой XAML в визуальное дерево

В коде приложения, реализующего часы, используется вспомогательные функции *root* и *use_xaml*. Рассмотрим их:

```
require 'Silverlight'

class Clock < SilverlightApplication
  use_xaml :type => Canvas

  def start
    d = Time.now()
    root.hour_animation.from    = from_angle  d.hour, 1, d.minute/2
    root.hour_animation.to      = to_angle    d.hour
    root.minute_animation.from  = from_angle  d.minute
    root.minute_animation.to    = to_angle    d.minute
    root.second_animation.from  = from_angle  d.second
    root.second_animation.to    = to_angle    d.second
  end

  def from_angle(time, divisor = 5, offset = 0)
    ((time / (12.0 * divisor)) * 360) + offset + 180
  end

  def to_angle(time)
    from_angle(time) + 360
  end
end

Clock.new.start
```

Обратите внимание на первую строку кода

```
require 'Silverlight'
```

это означает, что должен быть включен файл Silverlight.rb, потому класс нашего приложения будет наследоваться от него.

Затем объявляется класс *Clock* (Часы), производный от *SilverlightApplication* (определенный в Silverlight.rb):

```
class Clock < SilverlightApplication
```

Вызываем функцию *use_xaml* из *SilverlightApplication* и задаем, что используем XAML с базовым элементом <*Canvas*>. Это наследие Silverlight 1.0, в котором <*Canvas*> был единственным корневым элементом.

Далее переходим к функции *start*, которая вызывается функцией *use_xaml* по завершении инициализации. Она принимает текущее время с помощью функции *Time.now* и использует его для определения значений *from* (от) и *to* (до) для трех анимаций. Помните анимации *hour_animation*, *minute_animation* и *second_animation*? Теперь можно увидеть, как они используются. В Ruby вызов функции осуществляется путем указания ее имени, за которым следует пробел и список параметров через запятую. Таким образом, угол, представляющий текущее расположение часовой стрелки, получаем, вызывая *from_angle* и передавая в нее соответствующие параметры, которыми в данном случае являются текущее количество часов, 1, и количество минут, деленное на 2. Причина деления количества минут на два в том, что для трансформаций отсчет углов ведется от нижней точки окружности, т.е. здесь необходимо выполнить смещение на 180 градусов, следовательно, текущее показание делится на 2, чтобы «перевернуть» угол, предоставляемый значением *minute*.

Рассмотрим эту функцию:

```
def from_angle(time, divisor = 5, offset = 0)
   ((time / (12.0 * divisor)) * 360) + offset + 180
End
```

Возвращаясь к коду, можно увидеть, что начальное условие просто использует эту функцию для вычисления соответствующего угла трансформации, с которого будет начинаться анимация, и задает его в качестве значения свойства *from* анимации. Аналогично, значение окончания анимации (*to*) равно 360 градусам плюс текущее значение *from*. Таким образом, трансформация обеспечит прохождение стрелкой 360 градусов, начиная от исходного значения, за заданный промежуток времени (12 часов для часовой стрелки, 1 час для минутной и 1 минута для секундной стрелки).

Наконец, приложение начинается с создания нового экземпляра класса *Clock* и его запуска:

```
Clock.new.start
```

Таким образом, с помощью Ruby мы создали функциональное Silverlight-приложение – анимированные часы, которые отображают текущее время.

Заключение

В данной главе представлены динамические языки программирования и создано простое приложение *Hello World* с использованием IronPython, IronRuby и управляемого JavaScript. Вы увидели, как настраивать Visual Studio для работы с инструментом Chiron, применяемым для построения и управления выполнением динамических приложений.

Это всего лишь простая иллюстрация доступных возможностей, однако, этого достаточно, чтобы вдохновить вас на продолжение экспериментов с динамическими языками.

И на этом наше знакомство с Silverlight подошло к концу. Когда вы выбирали эту книгу, вероятно, вас интересовало, что такое Silverlight и что можно сделать с его помощью. Когда я писал ее, я не хотел создавать исчерпывающий энциклопедический справочник. Я просто хотел представить все замечательные возможности Silverlight в как можно более доступной форме, чтобы вы могли открыть любую главу, проработать ее и понять, что представляет собой эта технология, и захотели пойти дальше.

Надеюсь, вы получили такое же удовольствие от работы с этой книгой, какое испытывал я при ее написании!

Предметный указатель

Введение в Microsoft® Silverlight™ 3

ТРЕТЬЕ ИЗДАНИЕ

Ваш первый взгляд на инструменты и технологии создания приложений с использованием Silverlight 3

Начните применять Silverlight 3—кросс-платформенный, кросс-браузерный плагин для представления расширенных мультимедийных интерактивных приложений и пользовательских интерфейсов следующего поколения. Книга основана на знаниях команды разработчиков Microsoft Silverlight, процесс изучения построен с использованием практических упражнений, подробных указаний и примеров кода, чтобы вы сразу могли перейти к собственным разработкам.

Изучите как:

- Использовать XAML для отрисовки, масштабирования и анимации
- Воспользоваться мощью Microsoft .NET и удобством языков поддерживаемых Visual Studio®
- Применять разнообразные компоненты, создавая и расширяя свои собственные
- Экспериментировать с возможностями медиа
- Управлять приложениями Silverlight при помощи ASP.NET

Загрузите примеры к книге на C# и XAML
www.microsoft.com/rus/expression/resources

Об авторе

Лоуренс Морони (Laurence Moroney) является Ведущим экспертом по технологиям Microsoft, специализирующимся на Silverlight и User Experience. Имеет более чем десятилетний опыт разработки и реализации программного обеспечения и является автором около десятка книг по разнообразнейшим тематикам, среди которых Windows Presentation Foundation, Веб-разработка, безопасность и способность к взаимодействию. Кроме того, им написано более 150 статей для различных печатных и онлайн изданий и сделано множество докладов по этим темам на многих конференциях по всему миру.

ПОЛЕЗНЫЕ РЕСУРСЫ MICROSOFT®

Для дизайнеров
- Веб-сайт Expression
  www.microsoft.com/rus/expression
- Бюллетень Expression для дизайнеров
  www.microsoft.com/rus/expression/news-press/newsletter

Для веб-разработчиков
- Веб-технологии Microsoft
  www.microsoft.com/rus/web
- Центр разработки ASP.NET
  msdn.microsoft.com/ru-ru/asp.net

Для разработчиков
- Инструменты на русском языке
  www.visualstudio2008.ru
- Веб-сайт для разработчиков
  msdn.microsoft.com/ru-ru/

Подробнее см. на msdn.microsoft.com/ru-ru

Microsoft® Silverlight™

Microsoft®

Конференция веб-разработчиков
www.remix.ru